POLITECHNIKA BIAŁOSTOCKA

WYDZIAŁ INFORMATYKI

KATEDRA INFORMATYKI TEORETYCZNEJ

PRACA DYPLOMOWA MAGISTERSKA

TEMAT: LEARNING CANONICAL MODELS FROM DATA

WYKONAWCA: KRZYSZTOF NOWAK

.....

podpis

PROMOTOR: DR HAB. INŻ. MAREK JÓZEF DRUŻDŻEL

.....

podpis

BIAŁYSTOK 2013 r.

Karta dyplomowa

Politechnika Białostocka Wydział Informatyki	Studia stacjonarne studia II stopnia	Numer albumu studenta:75847
		Rok akademicki 2012/2013
Katedra Informatyki Teoretycznej		Kierunek studiów: informatyka Specjalność: Inżynieria Oprogramowania

Krzysztof Nowak

TEMAT PRACY: Learning canonical models from data

Zakres pracy:

1. Present Bayesian networks and the problem of knowledge engineering for Bayesian networks, with a special attention paid to learning numerical parameters from data
2. Describe so called canonical models and their significance
3. Propose a method for learning parameters of canonical models from data that uses all available records
4. Implement and test the proposed method on various data sets

.....	.....
Imię i nazwisko promotora podpis	Imię i nazwisko kierownika katedry - podpis

.....	.....	.....
Data wydania tematu pracy dyplomowej - podpis promotora	Regulaminowy termin złożenia pracy dyplomowej	Data złożenia pracy dyplomowej - potwierdzenie dziekanatu

.....	.....
Ocena promotora	Podpis promotora

.....	.....	.....
Imię i nazwisko recenzenta	Ocena recenzenta	Podpis recenzenta

Contents

Introduction

Probabilistic reasoning, among methodologies used within the domain of artificial intelligence, is recognized as one of the most promising foundations for machine learning techniques. Bayesian networks – one of the examples of a probabilistic modeling techniques – are widely acclaimed and used by both industrial and scientific communities. Apparatus for extracting information from ever–increasing amounts of data, was one of the key factors for the successes of companies like Google, IBM or Facebook. Nowadays, empirical data for many problem domains are freely accessible and, in many cases, growing at an exponential rate. There are fields of research, however, that are not privileged with an immensity of available data.

This document addresses the problem of eliciting probabilistic information from small samples of data. Areas where acquisition of empirical data is either costly or difficult, may benefit from any advancement in that regard. We will focus on learning parameters for the so–called *canonical models* of Bayesian networks as, in our opinion, currently available methods leave room for improvements.

First, we will start with an introduction to reasoning under uncertainty, Bayesian networks, and the canonical probabilistic models. Next, we will describe the problem of learning parameters for Bayesian networks, and for canonical gates in Bayesian networks. After the introductory part, we will propose a medium of communication between the domain of probabilistic events, and the field of linear algebra, in a form of the systems of simultaneous equations. Solution proposed by us relies on a technique for solving such systems, commonly known as Gauss–Jordan elimination. In order to acquaint the reader with the core method, we provide an extensive explanation supported by several examples.

Chapter 4 and further convey the theory and the experimental results behind several of the many possible variants of our method. We subject our solution to the varying state of the size of the learning sample, the number of variables or noise introduced by bending the constraints of the canonical model. Throughout the experiments, we compare our method against existing methods, and present the results of this comparison.

Last but not least, in Chapter 7 we describe the possible unexplored paths and improvements that may yield better understanding of the mechanisms embedded within the framework of canonical models, possibly improving the results even further.

1. Bayesian networks

In this chapter we will provide a brief introduction to Bayesian networks. We will start with describing the structure of a Bayesian network and the problem of determining causalities between variables. Second part of this chapter will be dedicated to description of the second component of a Bayesian network – numerical representation of the joint probability distribution (JPD).

1.1 Introduction to Bayesian networks

Representation of the interwoven probabilistic interactions between variables within a given problem domain is no trivial task. The term "Bayesian network" was first coined by Judea Pearl [1], who was among the first computer scientists to provide extensive research – both theoretical and experimental – on the topic.

The formal representation of a Bayesian network consists of the triple $(V, E, \mathbb{T})$, where

- V is a set of vertices depicting random variables associated with given problem's domain.
- E is a set of edges between pairs of vertices from V, capuring the probabilistic independences among the variables.
- $\mathbb{T}$ is a family of sets of conditional probability distributions. Each item from $\mathbb{T}$ is in one–to–one correspondence with some vertice from V.

Additionally, certain restrictions are imposed on graph (V, E): it must be **acyclic** and **directed**. Items from $\mathbb{T}$ are usually encoded as tables containing sets of conditional probability distributions. Random variables modelled by V can be either discreete or continuous, although more research was done on the former. Over the course of this document we will provide many examples of Bayesian networks, yet we will focus only on the variables with discreete domains.

1.2 Causality and Bayesian networks

Let us address the first aspect of Bayesian networks first – the cause and effect relationships among variables. Bayesian networks store that information in a form of a acyclic directed graph. Let us aid our explanation efforts by employing a frequently exemplified problem: modeling manifestation of cancer within a group of patients, based on basic information a general practitioner may gather. Let us assume that the variable of most interest will describe the occurrence of lung cancer, with other variables describing common environmental and genetics factors. Table 1.1 contains proposed discrete variables.

Variable	Description	Possible outcomes
LungCancer	Manifestation of lung cancer for given patient	{Malignant, Benign, None}
Smoking	Typical environmental factor – the habit of smoking tobacco products	{Yes, No}
SunlightExposure	Another environmental factor – how much sunlight exposure is the patient subjected to on a daily basis	{High, Medium, Low}
ChestPain	Occurrences of chest pains reported by the patient	{Yes, No}
History	The answer to the question: "Was any of your relatives diagnosed with any form of cancer?"	{YesLung, YesOther, No}
Asthma	Was the patient diagnosed with any form of asthma?	{Yes , No }

Table 1.1: Variables used for modeling the lung cancer problem

As of yet, we cannot be certain of any causal patterns within the given set of variables. An expert would probably propound the following relationship:

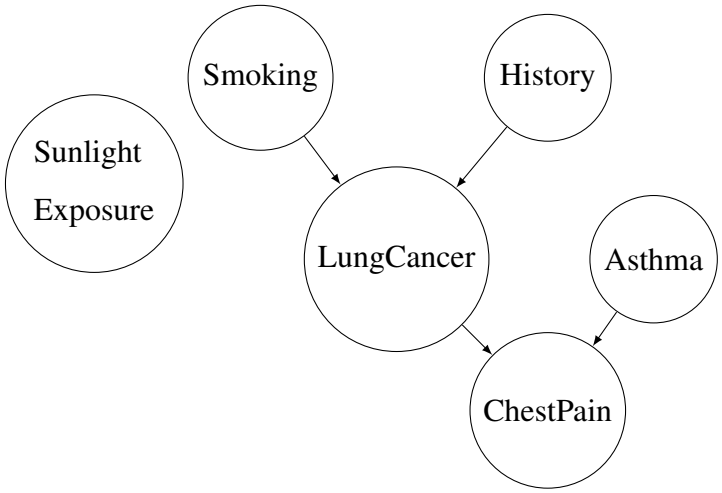

Figure 1.1: Graphical structure modeling the causalities of the problem.

Some conclusions can be drawn from the Figure 1.1:

- Smoking and history of cancer in the family, play a role in manifestation of cancer
- Lung cancer and asthma may induce chest pains

Our common sense would agree with the proposed structure, since the general knowledge about the problem is quite common. There exists a vast domain of problems where we do not have such convenience. In many cases, finding an expert for such problem is either very costly or simply impossible.

If we were to forget about our common sense for a while, any possible connection among the nodes above that satisfy the restrictions (graph has to be acyclic and directed), may be considered "valid". It would not necessarily describe the problem as it is, but without more information from the researchers in the field (which are often build and gathered over the course of many years), it is very hard to determine the correct connections solely from the database or the patient records. Problem of determining the structure of a Bayesian network is a difficult problem on its own. In our research, we aimed at improving the problem of parameter learning (which will be described in the next section), assuming that the structure of the network is given.

1.3 Representation of conditional probability distributions

In order to address the second component of a Bayesian network – representation of numerical parameters describing the conditional probability distributions between the variables – let us look closer at the graphical structure again (see 1.1). Since there is no direct connection between the nodes *SunlightExposure* and *LungCancer*, we do not assess any numerical value that would model conditional probability distribution linking both nodes. Similarly for pairs (*LungCancer*, *Asthma*) and (*LungCancer*, *ChestPain*). A node (in this case *LungCancer*) contains information on its conditional probability distribution conditional only of its parental nodes.

Bayesian networks contain information about direct impact of one variable on another in form of tables containing conditional probability distributions (CPT – Conditional Probability Table). Table 1.2 presents an example of such table. In cases where the node has no parents, we model the prior probability – a probability distribution according to which the state of given variable changes (see Table 1.3). Each node in a network is associated with either a CPT or a prior probability table.

Smoking	Yes			No		
History	YesLung	YesOther	None	YesLung	YesOther	None
Malignant	0.01	0.003	0.001	0.005	0.002	0.0001
Benign	0.02	0.006	0.001	0.001	0.003	0.0001
None	0.97	0.991	0.998	0.994	0.995	0.9998

Table 1.2: A possible CPT for the *LungCancer* node – independent parameters are highlighted in gray.

History	
YesLung	0.02
YesOther	0.13
No	0.85

Table 1.3: An example of prior probability table for the node History

Both graphical structure (Figure 1.1) and numerical values (tables such as Tables 1.2 and 1.3) define a Bayesian network. The formal structure on its own would be of limited use, if it were not for the algorithms allowing for inference within the network. As it is often the case in artificial intelligence, many problems can be thought of as general classification or prognostic problems. Diagnostic system allows for deducing the outcome of some variables, with incomplete knowledge for given scenario – e.g.,we would like to know the probability of a patient having lung cancer, knowing only the basic information on his tobacco smoking habits and family history.

2. Canonical probabilistic models

In this chapter we will describe so-called canonical models, which simplify many processes within Bayesian network, especially parameter learning. First we will briefly present a general framework of ICI (Independence of Causal Influences) models, along with the Noisy-OR and Noisy-MAX models. In the last section, we will compare both approaches (full CPT and Noisy-OR/MAX) with regard to learning parameters from data.

2.1 ICI models

ICI models are based on the assumption of independence of causal influences. A random variable may fit an ICI model if the mechanisms impacting it do not interact among each other. This simple restriction greatly simplifies elicitation of parameters from the data and the experts. The number of parameters required to define a probability distribution (CPT) is reduced, because some conditional probabilities can now be expressed as a function of a far smaller set of parameters. Figure 2.1 shows an example of such model.

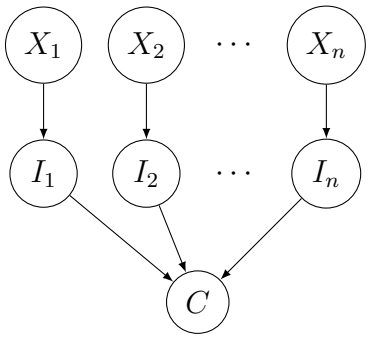

Figure 2.1: Structure of an ICI model

Additional auxilary nodes I_1 - I_n are called *inhibitors*. The independence of mechanisms is represented by a lack of edges among the inhibitor nodes I_1 to I_n.

2.2 Noisy-OR/MAX

Noisy-OR and Noisy-MAX models are cases of deterministic OR and deterministic MAX models [2, Section 3.1] applied to the ICI framework.

Deterministic models usually rely on a function that takes a set of input signals, from which the state of child node C is determined. The deterministic OR function makes it impossible for the child to be activated, if neither of the parent nodes is. Before we go any further, let us explain the meaning of the terms "activated" when referring to variable's state. Every variable has a special state indicating that the mechanism behind it is "off" or "absent," we call such state the "distinguished state." So the term "activated state" refers to one of the non-distinguished states. In order to be able to model real–life problems that touch matters other than death or taxes,[1] we have to embed the mechanism of uncertainty into the structure. This is done by introducing a separate layer of inhibitor nodes (introduced previously as nodes I_1 - I_n in Figure 2.1), which are activated based on their corresponding parent states (X_i) with a certain probability. Once that is done, the states of inhibitor nodes are used as input signals for the deterministic function. As it turns out, all we need to provide for to the model are the probabilities of every parental variable activating the child independently, that is when every variable (other than the one in question) is in its distinguished state.

Obtaining conditional probabilities for the scenarios other than the ones already described by the Noisy-OR parameters can be derived using the following equation:

$$P(\neg y|x) = \prod_{i \in I(x)} (1 - p_i) \ . \tag{2.1}$$

In our case the deterministic function is the OR function – the child is active when any of the inhibitors is active. Díez and Drużdżel have shown that deterministic OR is just a binary case for deterministic MAX function [2, Section 3.1]. For that reason, we can treat Noisy-OR as a binary case of the Noisy-MAX model. In further section we will occasionally use Noisy-OR/MAX interchangeably, knowing that general ideas resonate equally well for binary and non-binary variables.

[1] In one of the letters, Benjamin Franklin wrote the following to his friend, Jean-Batpiste Leroy: "Our new Constitution is now established, and has an appearance that promises permanency; but in this world nothing can be said to be certain, except death and taxes." – dated November 13th 1789.

Leaky-OR/MAX

As we had mentioned previously, deterministic OR function activates the child node only when at least one of the parents is in its non-distinguished states. This is not always the case in the real world – there are cases where absence of any signal from the parental causes still activates the child variable. Sometimes the problem does not allow for explicit modeling of each possible cause, either because it is not well understood, or because it would require large amount of additional variables that would have minimal impact on the child. In that case, unmodeled causes like that can be aggregated into a single node called *leak* (see Figure 4.1).

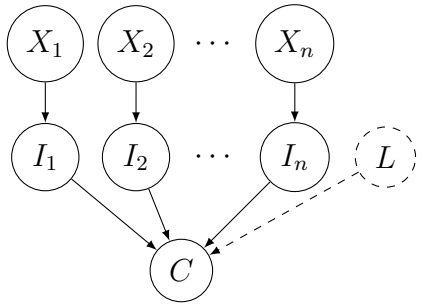

Figure 2.2: Structure of a Leaky-MAX model

Since leak is modeling mechanisms that we do not control or actively observe, it is assumed that every case where child is in one of its activated states, could be due to one of the causes aggregated within the leak variable. In order to incorporate that knowledge into previously derived Equation 2.1, we add the probability of leak to the product on the right hand side (Equation 2.2):

$$P(\neg y|x) = (1 - L) \cdot \prod_{i \in I(x)} (1 - p_i) \ . \tag{2.2}$$

Thanks to the restriction about independence of causal influences, we can obtain only some parameters (called Noisy-OR parameters), and compute the remaining ones. The Noisy-MAX table grows linearly with the number of parental nodes (see Table 2.1).

Parent	Smoking		History			Leak
State	Yes	No	YesLung	YesOther	None	
Malignant	0.01	0	0.005	0.002	0	0.01
Benign	0.01	0	0.004	0.003	0	0.01
None	0.98	1	0.991	0.995	1	0.98

Table 2.1: A possible leaky-MAX parameter table for the LungCancer node – independent parameters are highlighted gray

3. Parameter learning in Bayesian networks

In this chapter we will address a common problem of parameter learning, that exists within every type of model. In particular we will compare two previously presented models: CPT-based Bayesian network, and Noisy-MAX model. Efficient parameter learning is a problem with yet many unexplored paths and sub–problems. In our research, we try to analyze possible solutions for parameter learning, ignoring common problems, such as incomplete or continuous data. Learning is also based on certain assumptions that are to be explained in the subsequent chapters.

3.1 CPT-based model

It is not surprising that such accurate model comes with a cost. The problem of learning parameters from data was addressed previously by Drużdżel and Oniśko [3], Zagórecki and Voortman [4] and others e.g., [5, 6]. The main problem with defining and learning the CPT is the exponential growth of the table with the number of parents. The exact size of CPT and the number of independent parameters can be calculated as follows:

$$\text{CPTSize} = \text{Dim}(\text{Child}) \cdot \prod_{i=1}^{n} \text{Dim}(\text{Parent}_i) \ , \tag{3.1}$$

$$\text{IndependentParams} = (\text{Dim}(\text{Child}) - 1) \cdot \prod_{i=1}^{n} \text{Dim}(\text{Parent}_i) \ , \tag{3.2}$$

where $\text{Dim}(Q)$ is the number of possible outcomes for variable Q.

If we were to extend the lung cancer model (see Figure 1.1) to include more variables – let us say 10 parent variables that directly affect the *LungCancer* node – the size of the CPT would grow to $1,024$ columns (assuming that each of the 10 parents is binary). Note that the probability distribution in each of the columns, has to add up to 1. Because our *LungCancer* variable has three possibe outcomes, we require 2 values per each column, in order to fill out whole CPT (remaining third parameter is a supplemented to 1).

In real–life problems, it is not uncommon to find a model with thousands of variables, with many of them having up to a hundred of parent nodes. In such cases, the size of the model's CPTs can be huge. Not only the computational complexity of calculating inference within such network is difficult, but defining such model in the first place can be a headache

inducing experience. What we would have to ask the expert, are specific questions about conditional probability of every possible scenario that can occur within the problem domain.

This effort can be eased by learning the parameters from the datasets. In the basic form, it converges the learning efforts to sweeping through the database, and computing the required probabilities for each possible scenario using simple counting. This does not yet fix the problem. In order to have a meaningful probability distribution, let us settle on having at least 10 records describing given conditional probability distribution within a given CPT. Even if all cases were distributed uniformly, which they are rarely, we would require a sample record file of roughly $10 \cdot 2^n$ records (see Equation 3.2), and that is just for a binary case ($\text{Dim}(\text{Child}) = \text{Dim}(\text{Parent}_k) = 2$, for every k). In case of a model with 20 variables, this number increases to 1,048,576 columns. In almost every case, we would require much more than that to maintain a required 10 records per parameter ratio, since cases with unlike combinations of parents may rarely occur.

In practice, we have to use whatever database we have, and then verify the individual parameters with experts. Many obstacles may complicate the process even further: missing and incomplete data or continuous variables that need to be discretized first are fairly common in real datasets.

3.2 Noisy-MAX model

We had shown in previous section that using models based on the ICI model, reduces the number of parameters required to describe a node's CPT (see Table 2.1). This, of course, is tightly correlated with the parameter learning efforts for Noisy-MAX nodes. Noisy-OR/MAX models are widely used in situations where obtaining numerical parameters for CPT is difficult. The improvement in terms of a simpler elicitation of probability parameters and a smaller number of parameters that we have to provide (compared with full CPT) is significant.

The forumulas below calculate the sizes of Noisy-MAX table and the number of independent parameters for every table respectively:

$$\text{NoisyORSize} = \text{Dim}(\text{Child}) \cdot \sum_{i=1}^{n} \text{Dim}(\text{Parent}_i) + \text{Dim}(\text{Child}) \ , \tag{3.3}$$

$$\text{IndependentParams} = (\text{Dim}(\text{Child}) - 1) \cdot \sum_{i=1}^{n} (\text{Dim}(\text{Parent}_i) - 1) + (\text{Dim}(\text{Child}) - 1) \ . \tag{3.4}$$

3.3 Comparison

If we compare Equations 3.2 and 3.4, we can observe that the main difference is substituting the product on the right hand side of the equation by a sum. This has some interesting implications, since it reduces the growth of required parameters from exponential to linear. This directly translates to the efforts required for obtaining both sets of parameters (CPT and Noisy-MAX). We can observe however, that even though the second method does not require exploration of such vast universe of cases, it is prone to being less accurate. Two main reasons for that are the following:

1. The assumptions behind the Noisy-MAX model do not always hold in real–life problems

2. Prior probabilities of parental nodes dictate the frequency with which the Noisy-MAX parameters occur within the data

Let us start with addresing the first problem. As it was described in Section 2.2, Noisy-MAX models are based on ICI models. ICI models rely on certain relationship between parental nodes being true, namely: mechanism behind parental variables must produce the effect on child **independently**. If this is not the case however, trying to learn Noisy-MAX parameters using standard approach may yield poor results (after all, we are assuming certain conditions that do not hold).

As for the second reason against Noisy-MAX models – I will explain it through an example. Let us assume that we are working on a network consisting of 3 binary parents and one child node (Noisy-OR gate) (see Figure 3.1). Let us also assume that the prior probability table of each parent X_1, X_2 and X_3 is as presented in the Table 3.1.

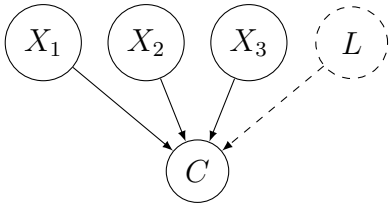

Figure 3.1: Simple Bayesian network with 3 parents ($X_1 - X_3$), child node (C) and implicit leak node (L)

State	Probability
1	0.8
0	0.2

Table 3.1: Prior probability table for nodes X_1, X_2 and X_3

Out of the Table 3.1, we can derive the Table 3.2, containing joint probabilities over each possible scenario of parents' outcomes.

(x_1, x_2, x_3)	P(x_1, x_2, x_3)
(1, 1, 1)	0.512
(1, 1, 0)	0.128
(1, 0, 1)	0.128
(0, 1, 1)	0.128
(1, 0, 0)	0.032
(0, 1, 0)	0.032
(0, 0, 1)	0.032
(0, 0, 0)	0.008

Table 3.2: Possible outcomes of parent states along with their corresponding probability – combinations defining the Noisy-MAX parameters are highlighted in gray

Under ideal conditions, any means for eliciting the parameters required by the Noisy-MAX would yield the correct values. However, this is never the case when learning parameters from data. If we were to learn the parameters from a database consisting of 1.000

records, the CPT parameter corresponding to the case (x_1, x_2, x_3) = $(0, 1, 1)$ would be represented by more records (128) than all parameters for the Noisy-MAX gate (104). Weak representation in sample data results in high variance of error for the learnt parameters.

4. A new method for learning parameters of canonical models

This chapter merges the concepts of solving systems of linear equations with the fundamental part of the Noisy-MAX gates – definition of conditional probability based on Noisy-MAX parameters (Equation 2.2). First we will propose an interpretation of probabilistic events as a system of linear equations. Later we will describe a method for solving over-determined and under-determined systems of equations, using Gauss-Jordan elimination. Finally we will describe the core method itself, and possible variants that rely on it. The method itself is based on elementary linear algebra, and the idea behind it is quite simple.

4.1 Interpreting probabilistic events as systems of equations

In order to show the relationship between probabilistic events and systems of linear equations, we will rely on an example. Let us assume the following structure of a Noisy-MAX network:

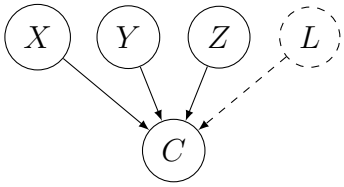

Figure 4.1: Structure of a Noisy-MAX model

Let us also assume that the parents X and Y are binary, while parent Z has 3 possible outcomes. The space of all possible combinations of parent states, along with the conditional probability of activating the child node is as follows (each state is lower cased version of variable's name with an index describing the state number):

$$\left[\begin{array}{ccc} X & Y & Z \\ \hline x_1 & y_1 & z_1 \\ x_1 & y_1 & z_2 \\ x_1 & y_1 & z_3 \\ x_1 & y_2 & z_1 \\ x_1 & y_2 & z_2 \\ x_1 & y_2 & z_3 \\ x_2 & y_1 & z_1 \\ x_2 & y_1 & z_2 \\ x_2 & y_1 & z_3 \\ x_2 & y_2 & z_1 \\ x_2 & y_2 & z_2 \\ x_2 & y_2 & z_3 \end{array}\right] \sim \textcircled{1} \sim \left[\begin{array}{c} P(C = c_1 | X = x_i, Y = y_j, Z = z_k) \\ \hline (\neg p_{x_1}) \cdot (\neg p_{y_1}) \cdot (\neg p_{z_1}) \cdot (\neg L) = (\neg v_1) \\ (\neg p_{x_1}) \cdot (\neg p_{y_1}) \cdot (\neg p_{z_2}) \cdot (\neg L) = (\neg v_2) \\ (\neg p_{x_1}) \cdot (\neg p_{y_1}) \cdot (\neg L) = (\neg v_3) \\ (\neg p_{x_1}) \cdot (\neg p_{z_1}) \cdot (\neg L) = (\neg v_4) \\ (\neg p_{x_1}) \cdot (\neg p_{z_2}) \cdot (\neg L) = (\neg v_5) \\ (\neg p_{x_1}) \cdot (\neg L) = (\neg v_6) \\ (\neg p_{y_1}) \cdot (\neg p_{z_1}) \cdot (\neg L) = (\neg v_7) \\ (\neg p_{y_1}) \cdot (\neg p_{z_2}) \cdot (\neg L) = (\neg v_8) \\ (\neg p_{y_1}) \cdot (\neg L) = (\neg v_9) \\ (\neg p_{z_1}) \cdot (\neg L) = (\neg v_{10}) \\ (\neg p_{z_2}) \cdot (\neg L) = (\neg v_{11}) \\ (\neg L) = (\neg v_{12}) \end{array}\right] \sim \textcircled{2} \ldots \tag{4.1}$$

$$\ldots \textcircled{2} \sim \left[\begin{array}{c} log_b(P(C = c_1 | X = x_i, Y = y_j, Z = z_k)) \\ \hline \log_b(\neg p_{x_1}) + \log_b(\neg p_{y_1}) + \log_b(\neg p_{z_1}) + \log_b(\neg L) = log_b(\neg v_1) \\ \log_b(\neg p_{x_1}) + \log_b(\neg p_{y_1}) + \log_b(\neg p_{z_2}) + \log_b(\neg L) = log_b(\neg v_2) \\ \log_b(\neg p_{x_1}) + \log_b(\neg p_{y_1}) + \log_b(\neg L) = log_b(\neg v_3) \\ \log_b(\neg p_{x_1}) + \log_b(\neg p_{z_1}) + \log_b(\neg L) = log_b(\neg v_4) \\ \log_b(\neg p_{x_1}) + \log_b(\neg p_{z_2}) + \log_b(\neg L) = log_b(\neg v_5) \\ \log_b(\neg p_{x_1}) + \log_b(\neg L) = log_b(\neg v_6) \\ \log_b(\neg p_{y_1}) + \log_b(\neg p_{z_1}) + \log_b(\neg L) = log_b(\neg v_7) \\ \log_b(\neg p_{y_1}) + \log_b(\neg p_{z_2}) + \log_b(\neg L) = log_b(\neg v_8) \\ \log_b(\neg p_{y_1}) + \log_b(\neg L) = log_b(\neg v_9) \\ \log_b(\neg p_{z_1}) + \log_b(\neg L) = log_b(\neg v_{10}) \\ \log_b(\neg p_{z_2}) + \log_b(\neg L) = log_b(\neg v_{11}) \\ \log_b(\neg L) = log_b(\neg v_{12}) \end{array}\right] \sim \textcircled{3} \ldots \tag{4.2}$$

$$\dots (3) \sim \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} log_b(\neg x_1) & log_b(\neg y_1) & log_b(\neg z_1) & log_b(\neg z_2) & log_b(\neg L) \end{bmatrix} = \begin{bmatrix} \neg v_1 \\ \neg v_2 \\ \neg v_3 \\ \neg v_4 \\ \neg v_5 \\ \neg v_6 \\ \neg v_7 \\ \neg v_8 \\ \neg v_9 \\ \neg v_{10} \\ \neg v_{11} \\ \neg v_{12} \end{bmatrix} \tag{4.3}$$

Equivalences 4.1, 4.2 and 4.3 show a transition from the space of all possible combinations of parents' states to a system of simultaneous linear equations. Intermediate steps to achieve that are denoted as (1), (2) and (3).

Step (1): First step derives directly from the Equation 2.2. We subject each scenario of parent states to the equation above. This results in a set of product-equations that describe the probability that a given combination of parents activates the child node.

Step (2): In this step, we reduce the system of product equations to a system of linear equations, by taking a logarithm of both sides of each equation.

Step (3): Transition in this step is simply writing out the system of equations in a matrix notation.

We have shown that any combination of parent states can be represented as a numerical vector describing exponents of each product-equation component.

4.2 Methods for solving systems of linear equations

Some approaches to solving a self-contained system of linear equations – such as Cramer's rule – require enough constrains to ensure an independent solution, without it being overdetermined at the same time. Numerical methods are also an option, but they may

be difficult to control, and could add additional approximations to the solution.

Overdetermined systems of equation contain more equations than unknowns. Similarly, underdetermined systems of equations do not contain sufficiently many linearly independent vectors to find a solution. In all cases that we will work with later, the set of available equations is greater than the number of unknowns, so in most cases we will work with overdetermined systems of equations. It is also worth mentioning that overdetermined systems of equation do not always guarantee that every unknown can be solved independently. This invokes additional restriction on the equations – there must be at least n linearly independent equations among them.

We can avoid this problem entirely, by using Gauss–Jordan elimination. Gauss–Jordan elimination works equally well with work with both overdetermined and underdetermined systems of equations. The order of equations is also taken into account, so in cases of contradictions, certain combinations are preferred to others.

4.2.1 Gauss–Jordan elimination

Gauss–Jordan elimination is well defined as an algorithm for solving systems of linear equations, that is systems of the form:

$$\begin{cases} c_{11} \cdot x_1 + c_{21} \cdot x_2 + \cdots + c_{n1} x_n = b_1 \\ c_{12} \cdot x_1 + c_{22} \cdot x_2 + \cdots + c_{n2} x_n = b_2 \\ \vdots \\ c_{1m} \cdot x_1 + c_{2m} \cdot x_2 + \cdots + c_{nm} x_n = b_m, \end{cases} \tag{4.4}$$

or by using the matrix-notation:

$$\begin{bmatrix} c_{11} & c_{21} & \cdots & c_{n1} \\ c_{12} & c_{22} & \cdots & c_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1m} & c_{2m} & \cdots & c_{nm} \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}. \tag{4.5}$$

Let us propose a simple equation set presented in a standard matrix form $A \cdot X = b$.

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}. \tag{4.6}$$

We will use symbols for vector b (vector of constants), so we can keep track of the elementary row operations. Turning that into an augmented matrix $[A|b]$ yields

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right]. \tag{4.7}$$

Before we start, notice that this equation set may be contradictory if we were to use real numbers – x_3 can be calculated as a linear combination of two first rows ($x_3 = b_1 - b_2$) or by taking the fourth row as a solution ($x_3 = b_4$). It is plausible that substituting b_1, b_2 and b_3 with real values calculated from the database, would lead to x_3 being equal to two different values. Additionally, the equation set is underdetermined – there is not enough information about x_1 and x_2 to solve them independently.

We will now perform Gauss–Jordan elimination steps in order to show that certain properties we care about (such as preserving preference of equations determined by their order) are taken into account.

We will distinguish pivot elements with colors red (currently selected pivot element) and blue (previous pivot elements). During the algorithm, we will prefer the topmost pivot element in given column – this way we assure that topmost equations will be more significant.

1. We choose the first pivot element (in red), and use it to zero-out remaining coefficients in the first column:

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right] \begin{array}{l} \\ r_2 = r_2 - r_1 \\ \\ \\ \end{array} \sim \left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 0 & 0 & -1 & 0 & b_2 - b_1 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right]. \tag{4.8}$$

2. We select the second pivot element – note that no two pivot elements can share the same row. The first non-zero element that satisfies this condition is coefficient of x_3 in the second row:

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 0 & 0 & -1 & 0 & b_2 - b_1 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right] r_2 = r_2 \cdot (-1) \sim \left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 0 & 0 & 1 & 0 & b_1 - b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right], \tag{4.9}$$

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 0 & 0 & 1 & 0 & b_1 - b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right] \begin{array}{l} r_1 = r_1 - r_2 \\ \\ \\ r_4 = r_4 - r_2 \end{array} \sim \left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & b_1 - (b_1 - b_2) \\ 0 & 0 & 1 & 0 & b_1 - b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 0 & 0 & b_4 - (b_1 - b_2) \end{array}\right]. \tag{4.10}$$

3. The last pivot element is going to be coefficient at x_4 in the third row. Since the remaining coefficients are all zeros in the fourth column, no changes are made. We can simplify the values in new vector b. Notice that fourth row is a zero-vector – we can eliminate that from the equation set:

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & b_1 - (b_1 - b_2) \\ 0 & 0 & 1 & 0 & b_1 - b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 0 & 0 & b_4 - (b_1 - b_2) \end{array}\right] \sim \left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & b_2 \\ 0 & 0 & 1 & 0 & b_1 - b_2 \\ 0 & 0 & 0 & 1 & b_3 \end{array}\right]. \tag{4.11}$$

Let us compare our end-result with the initial matrix:

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right]. \tag{4.12}$$

As we can see, x_3 was calculated using the first and the second row $(b_1 - b_2)$. Let us see what happens after we move the third row on the top position, indicating that our preferred ordering of equations changed. (We expect now to calculate x_3 solely by first row).

1. We choose the our first pivot element, and use it to zero-out remaining coefficients in first column:

$$\left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 1 & 0 & b_2 \\ 1 & 1 & 0 & 0 & b_3 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right] \begin{array}{l} \\ \\ r_3 = r_3 - r_2 \\ \\ \end{array} \sim \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 1 & 0 & b_2 \\ 0 & 0 & -1 & 0 & b_3 - b_2 \\ 0 & 0 & 0 & 1 & b_1 \end{array}\right]. \tag{4.13}$$

2. Again, no candidate for pivot element in the second column, coefficient at x_3 in the first row is the next pivot element:

$$\left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 1 & 0 & b_2 \\ 0 & 0 & -1 & 0 & b_3 - b_2 \\ 0 & 0 & 0 & 1 & b_1 \end{array}\right] \begin{array}{l} \\ r_2 = r_2 - r_1 \\ r_3 = r_3 + r_1 \\ \\ \end{array} \sim \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 - b_1 \\ 0 & 0 & 0 & 0 & b_3 - b_2 + b_1 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right]. \tag{4.14}$$

3. Last item fit for a pivot element is a coefficient at x_4 in the fourth row. After getting rid of zero vectors, we achieve the following reduced row echelon matrix form:

$$\left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 - b_1 \\ 0 & 0 & 0 & 0 & b_3 - b_2 + b_1 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right] \sim \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 - b_1 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right]. \tag{4.15}$$

As expected, x_3 was calculated using the most preferred set of equations, as dictated by their order. What this method does not take into account is the relative weight of each row. As of yet, all we could rely on was simple ordering of equations, without using the information about quantity or frequency of each type of equation in our learning set.

4.2.2 Properties of zero vectors

In this section, we will describe how solving the equation set using one order does not prevent us from reproducing other solutions for a given parameter. As we saw previously, different order of equations may lead to different outcomes for certain parameters. This variety comes from the contradictions in the equation set.

Definition. Zero vector

We will use the term "zero vector" to describe the vectors of the form

$$\left[\begin{array}{cccc|c} 0 & 0 & \dots & 0 & M(b) \end{array}\right], \tag{4.16}$$

where $M(b)$ is a linear combination of absolute terms (vector b).

Zero vectors that emerge during Gauss–Jordan elimination contain information about other possible solutions for a given value. Instead of removing them in the process, we can store them, and utilize them later. Let us see how previous example holds to this theory:

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \end{array}\right] \sim \left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & b_1-(b_1-b_2) \\ 0 & 0 & 1 & 0 & b_1-b_2 \\ 0 & 0 & 0 & 1 & b_3 \\ 0 & 0 & 0 & 0 & b_4-(b_1-b_2) \end{array}\right] \sim \left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & b_2 \\ 0 & 0 & 1 & 0 & b_1-b_2 \\ 0 & 0 & 0 & 1 & b_3 \end{array}\right]. \tag{4.17}$$

This particular order of equation leads to x_3 being calculated from two top-most equations in a set. If we add our final solution for x_3 (vector in red), to the zero vector we ought to remove in a last step of our algorithm (vector in blue), we obtain previously abandoned solution:

$$\left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1-b_2 \end{array}\right] + \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & b_4-(b_1-b_2) \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_4 \end{array}\right]. \tag{4.18}$$

Using the same method we can start from the solution obtained after rearranging the order of equations in the initial matrix:

$$\left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 1 & 0 & b_2 \\ 1 & 1 & 0 & 0 & b_3 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right] \sim \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2-b_1 \\ 0 & 0 & 0 & 0 & b_3-b_2+b_1 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right] \sim \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \\ 1 & 1 & 0 & 0 & b_2-b_1 \\ 0 & 0 & 0 & 1 & b_4 \end{array}\right]. \tag{4.19}$$

Linear combination of two vectors again yields a different solution:

$$\left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_1 \end{array}\right] + (-1) \cdot \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & b_3-b_2+b_1 \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & 1 & 0 & b_2-b_3 \end{array}\right]. \tag{4.20}$$

Let us look at the example of the equation set with multiple zero vectors:

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 1 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 1 & 1 & 0 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \\ 1 & 0 & 0 & 0 & b_5 \\ 1 & 0 & 0 & 1 & b_6 \end{array}\right] . \tag{4.21}$$

This equation set is overconstrained – we can expect to obtain at least two zero vectors after the Gauss–Jordan elimination steps:

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 1 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 1 & 1 & 0 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \\ 1 & 0 & 0 & 0 & b_5 \\ 1 & 0 & 0 & 1 & b_6 \end{array}\right] \sim \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & b_2 - b_3 + b_4 \\ 0 & 0 & 0 & 1 & b_1 - b_2 \\ 0 & 1 & 0 & 0 & b_3 - b_4 \\ 0 & 0 & 1 & 0 & b_4 \\ 0 & 0 & 0 & 0 & -b_2 + b_3 - b_4 + b_5 \\ 0 & 0 & 0 & 0 & -b_1 + b_3 - b_4 + b_6 \end{array}\right] \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} . \tag{4.22}$$

Let us mark each vector in a reduced row echelon form (Equation 4.22) as $v_1 \dots v_6$. As we have shown previously, we can use zero vectors to obtain different solutions to parameters $x_1 \dots x_4$, for example:

	combination	value
$[1 \quad 0 \quad 0 \quad 0]_1$	v_1	$b_2 - b_3 + b_4$
$[1 \quad 0 \quad 0 \quad 0]_2$	$v_1 + v_5$	b_5
$[1 \quad 0 \quad 0 \quad 0]_3$	$v_1 + v_6$	$-b_1 + b_2 + b_6$
$[0 \quad 1 \quad 0 \quad 0]_1$	v_3	$b_3 - b_4$
$[0 \quad 1 \quad 0 \quad 0]_2$	$v_3 - v_5$	$b_2 - b_5$
$[0 \quad 1 \quad 0 \quad 0]_3$	$v_3 - v_6$	$b_1 - b_6$
$\dots$	$\dots$	$\dots$
$[0 \quad 0 \quad 0 \quad 1]_1$	v_2	$b_1 - b_2$
$[0 \quad 0 \quad 0 \quad 1]_2$	$v_2 + v_6 - v_5$	$b_6 - b_5$
$\dots$	$\dots$	$\dots$

(4.23)

Notice that the second solution for x_4 (in red above) requires two zero vectors to find an efficient solution.

We would like to propose a theorem describing relationship of possible solutions with zero vectors in the reduced row echelon matrix form.

Zero–vector theorem

Every possible solution for a given parameter q can be obtained as a linear combination of a solution vector from Gauss–Jordan elimination, and the zero vectors, i.e.,

$$\begin{aligned} \forall_{s \in S} \exists_{a \in V} \; s &= [1, a_1, a_2, \ldots, a_n] \cdot [s_0, z_1, z_2, \ldots, z_n] = \\ &= s_0 + a_1 \cdot z_1 + a_2 \cdot z_2 + \cdots + a_n \cdot z_n \end{aligned} \tag{4.24}$$

where $s \in S$ is a solution vector s for parameter q from a space of all possible solutions S

$a \in V$ is the vector of coefficients from a vector space V over a field $\mathbb{R}$

s_0 is the first solution (also a vector over a field $\mathbb{R}$) for given parameter, as obtained from Gauss–Jordan elimination

z_i is the i-th zero vector obtained from Gauss–Jordan elimination ($i \in 1 \ldots n$).

n is the number of zero vectors in reduced row echelon form.

Proof

If the equation set is self–contained, each parameter has an independent solution. Since in that case there are no zero vectors, $n = 0 => s = [1] \cdot [s_0] = s_0$.

Similar case would emerge when the equation set is strictly underdetermined (not every parameter has an independent solution, but no zero vectors appear in reduced row echelon form either). Third case would be equation sets with over-constrainments, which are of our interest here since they produce zero vectors after Gauss–Jordan elimination.

First, let us define two terms that we will later use:

Definition. Linear combination of equation set

Linear combination of the equation set can be interpreted as a function

$$f : \mathbb{M}_{m\times n} \to \mathbb{M}_{m\times n}, \tag{4.25}$$

where $\mathbb{M}_{m\times n}$ is a space of matrices of size $m \times n$.

Additionally every such function f is equivalent to left multiplication by some matrix F, that is

$$\forall_f \forall_{A_0 \in \mathbb{M}_{m\times n}} \exists_{F \in \mathbb{M}_{m\times m}} \; f(A_0) = F \cdot A_0. \tag{4.26}$$

Example: Equivalence relation 4.22 could also be written as an equation:

$$\left[\begin{array}{cccccc} 0 & 1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \end{array}\right] \cdot \left[\begin{array}{cccc|c} 1 & 1 & 0 & 1 & b_1 \\ 1 & 1 & 0 & 0 & b_2 \\ 0 & 1 & 1 & 0 & b_3 \\ 0 & 0 & 1 & 0 & b_4 \\ 1 & 0 & 0 & 0 & b_5 \\ 1 & 0 & 0 & 1 & b_6 \end{array}\right] = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & b_2 - b_3 + b_4 \\ 0 & 0 & 0 & 1 & b_1 - b_2 \\ 0 & 1 & 0 & 0 & b_3 - b_4 \\ 0 & 0 & 1 & 0 & b_4 \\ 0 & 0 & 0 & 0 & -b_2 + b_3 - b_4 + b_5 \\ 0 & 0 & 0 & 0 & -b_1 + b_3 - b_4 + b_6 \end{array}\right] \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \tag{4.27}$$

in which case the leftmost matrix would be our linear combination of the Gauss–Jordan elimination algorithm.

Definition. Solution vector

Solution vector is a single row in a matrix (usually obtained by linear combination of the equation set), directly solving given parameter x_k (vector $[\ 0\ \ldots\ 0\ 1\ 0\ \ldots 0 \mid b\ ]$ with "1" at the k-th place, and some constant term b in its augmented form).

Example: Row v_2 in equation 4.27 unambiguously gives solution to parameter x_4, thus vector $[\ 0\ 0\ 0\ 1 \mid b_1 - b_2\ ]$ is the solution vector of x_4.

We will now prove the theorem in question.

Let $\boldsymbol{A}$ be the original equation set, and $\boldsymbol{B}$ – the equation set after Gauss–Jordan elimination (reduced row echelon form). Let us say that given parameter x_m can be calculated using at least two different linear combinations of vectors from $\boldsymbol{A}$. Let us call

these L_0 and L_k, where L_0 is linear combination equivalent to Gauss–Jordan elimination ($\boldsymbol{B} = L_0 \cdot \boldsymbol{A}$). Let us assume that s_0 is the solution vector for parameter x_m as obtained from Gauss–Jordan elimination (that is $s_0 \in \boldsymbol{B}$)

Facts:

1. s_0 is a solution vector of x_m in in $\boldsymbol{B}$.
2. s_k is a solution vector of x_m in $L_k \cdot \boldsymbol{A}$, but it is not a vector in $\boldsymbol{B}$.

We can show that s_k can also be obtained as linear combination of vectors from $\boldsymbol{B}$ using only s_0 and the zero vectors, by splitting the theorem into two parts:

1. Vector s_k can be obtained as a linear combination of vectors from $\boldsymbol{B}$:
   Since L_0 is determined by Gauss–Jordan elimination, which in turn uses only elementary row operations, L_0 is an invertible matrix. Thus L_0^{-1} exists. Because $L_0^{-1} \cdot L_0 = \mathrm{Id}$, and matrix multiplication is associative, we can apply the following:

   $$s_k \in L_k \cdot \boldsymbol{A} \Rightarrow s_k \in L_k \cdot (L_0^{-1} \cdot L_0) \cdot \boldsymbol{A} \Rightarrow s_k \in L_k \cdot L_0^{-1} \cdot \boldsymbol{B}. \tag{4.28}$$

   Because s_k is a vector in $L_k \cdot \boldsymbol{A}$, s_k is also a vector in $L_k \cdot L_0^{-1} \cdot \boldsymbol{B}$. In that case, we can use a linear combination $L_k \cdot L_0^{-1}$ to go from solution s_0 to s_k.

2. Vector s_k in $L_k \cdot L_0^{-1} \cdot \boldsymbol{B}$ is a linear combination of vectors no other than s_0 and the zero vectors in $\boldsymbol{B}$:
   Because $\boldsymbol{B}$ is a reduced row echelon form, the following property holds: no two non-zero coefficients in reduced row echelon form share the same column.

   Let us assume that vector s_k is calculated using two non-zero vectors in its linear combination from $\boldsymbol{B}$ to $L_k \cdot \boldsymbol{A}$. In that case, s_0 has to appear in a linear combination with a non-zero coefficient since no other non-zero vector can produce a 1 in m-th column in s_k. Additionally, any linear combination involving any two non-zero vectors from $\boldsymbol{B}$ with both coefficients other than 0 will produce a vector with at least two coefficients other than 0. Such vector would not be a solution vector, which s_k is. We obtain, thus, a contradiction, which proves the theorem. □

Finding such linear combination is not a trivial task. The solution space is infinite, and virtually any linear combination of zero vectors can be added to any non-zero vector, giving us an valid solution (although the combination would be very inefficient in most cases)

4.3 Learning the leak parameter from data

Learning parameters for the Noisy-MAX models from the data is different than obtaining them from experts. There are many differences in that regard, but the main point that really impacts our approach to learning is the special significance of leak. In this chapter we will describe two different definitions of leak, and present the problem of learning leak from data.

4.3.1 Eliciting leak parameter

First, let us discuss the most common approach to eliciting the leak parameter from data. Since leak is slightly different from ordinary parameters, we have to approach this endeavour with high caution. Leak, when not expressed explicitly in the data, is represented by the conditional probability of the child node being activated, given that all of the parent nodes are in their distinguished states. Let us assume the network with the structure as

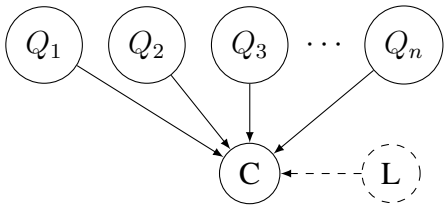

Figure 4.2: Structure of simple Bayesian Network

	Q_1	Q_2	Q_3	$\ldots$	Q_n
$P(Q_k = +)$	p_1	p_2	p_3	$\ldots$	p_n
$P(Q_k = -)$	$\neg p_1$	$\neg p_2$	$\neg p_3$	$\ldots$	$\neg p_n$

Table 4.1: Aggregated table of prior probabilities of each parent node

presented in Figure 4.2. Additionally, let us assume that each parent is binary, and their

prior probability distributions are as presented in Table 4.1. In that case, incidence of records within the data file describing the leak directly is equal to:

$$\prod_{i=1}^{n}(\neg p_i). \tag{4.29}$$

A similar product can, of course, be derived for any combination of parents' states. As we had shown previously, in case of other parameters we can aid our efforts of solving parameters by means of linear combination of vectors and linear algebra. We would like to achieve similar freedom in solving for the leak as well. This is especially important if parameters $p_1 \cdots p_n$ are relatively high, thus, favoring non-distinguished states to occur more often. Since in such cases the number of records describing the leak can be relatively small, we can try to express the leak as one of the explicit parameters. Let us compare two approaches to interpreting the parameters of the model due to Díez and Henrion respectively [2, Section 4.1 and 4.2]:

$$P(\neg y|x) = (\neg p_L) \cdot \prod_{i \in I(x)} (\neg p_i), \tag{4.30}$$

$$P(\neg y|x) = (\neg p_L) \cdot \prod_{i \in I(x)} \frac{\neg p_i'}{\neg p_L}. \tag{4.31}$$

Parameter p_k is sometimes called a *net parameter*, while parameter p_k' is called a *compound parameter*. Noisy-MAX gate, as defined in this document, requires *net parameters*. However, transition between p_k and p_k' for any k is quite straightforward:

$$\neg p_k = \frac{\neg p_k'}{\neg p_L}. \tag{4.32}$$

First definition is used when we are eliciting parameters from an expert. Since the main concern of our research is learning from data, Henrion's definition (Equation 4.31) seems to fit to our purpose better.

Let us see how we would approach adding leak to the data:

#	Q_1	Q_2	Q_3	Q_4	Leak	$P(C = True\|Q)$
1	1	1	0	0	-1	c_1
2	0	1	1	1	-2	c_2
3	1	1	1	0	-2	c_3
4	0	0	1	0	0	c_4
5	0	0	0	0	1	c_5

Table 4.2: Example of data supplemented with an explicit leak column.

A negative value in the leak column (see Table 4.2), indicates positive exponent in the denominator of the product (see Equation 4.31). This is also the result of taking the logarithm of both sides of the equation – coefficient at $\log_b(\neg p_L)$ is negative:

$$\log_b(\neg p_L \cdot \frac{\neg p_1}{\neg p_L} \cdot \ldots \cdot \frac{\neg p_n}{\neg p_L}) = \log_b(\neg p_1) + \ldots + \log_b(\neg p_n) - (n-1) \cdot \log_b(\neg p_L) \quad (4.33)$$

Solving the supplemented system of equations will provide us with the *compound parameters* and the leak parameter. In order to obtain the *net parameters*, we apply Equation 4.32.

Let us notice that when we model the records within the data by means of a system of equations, every parameter p_i has binary coefficient, except for the leak. For that reason the leak requires an especially high accuracy when learnt from the data. Because it takes part in almost every equation (except for the solution vectors) with a relatively high exponent (exponent of leak grows linearly with the number of present causes), it may introduce error to every parameter p_i.

4.4 Different approaches to solving parameters

Since our method enables us to explore different solutions for each parameter (in case of overdetermined systems), we propose several criteria by which we choose the final value for a given parameter. These can rely on choosing the best solution out of all that are possible, or combining several best choices together. These can be divided into two classes of methods:

1. **Quality–based methods.**

   This class of methods is focusing on choosing a particular permutation of the initial equations in the equation set, so that after the Gauss–Jordan elimination, we obtain good candidates for solutions to the Noisy-MAX parameters right away. For that reason, they do not make use of zero vectors and are relying on the information contained within a given ordering of the equations.

2. **Exploration of zero vectors.**

   This class of methods is based on the zero–vector theorem that was proved in previous chapter. In this case, initial solution to the parameter is also important, in order to simplify the process of exploration of zero vectors. For that reason, this class of methods is relying on quality–based approach.

We will support the theory with a simple example, based on the network presented in Figure 4.3. Let us assume that all nodes are binary, and the sampled data are as presented in Table 4.3.

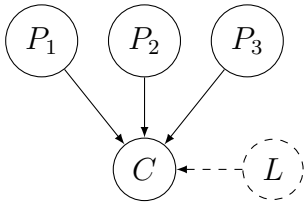

Figure 4.3: Structure of a Bayesian network with 3 parents and 1 child.

#	p_1	p_2	p_3	p_L	Cnt($C = +, \mathbf{P}$)	Cnt($\mathbf{P}$)
1	0	0	0	1	1	36
2	0	0	1	0	4	32
3	0	1	0	0	2	17
4	0	1	1	-1	4	16
5	1	0	0	0	300	330
6	1	0	1	-1	249	263
7	1	1	0	-1	157	167
8	1	1	1	-2	134	139

Table 4.3: Possible data for the network in Figure 4.3, generated from a sample of 1,000 records.

We we will use the shorthand notation "Cnt($\mathbf{P}$)" to describe the number of records in which parents' states are in some specific combination: $(P_1 = p_1, P_2 = p_2, P_3 = p_3)$. Similarly, we will denote the number of records that meet the condition $(C = +, P_1 = p_1, P_2 = p_2, P_3 = p_3)$ by Cnt($C = +, \mathbf{P}$) – notation $C = +$ describes the child being in its non-distinguished state. Before we start describing methods that are based on Gauss–Jordan elimination, let us introduce a reference algorithm to which we will compare our solution in later chapters.

4.4.1 Simple Counting

Simple Counting method reduces to eliciting Noisy–MAX parameters from the solution vectors within the data. Conditional probability is then calculated by comparing the numbers of records that activated the child, to the number records that did not. In order to compute the Noisy–MAX parameters from the data in Table 4.3, Counting method would simply obtain them through vectors $1, 2, 3$, and 5.

4.4.2 Fitness–based methods

Let us focus on the first class of solutions first. In general, fitness–based methods revolve around the idea that we can assign a "quality" measure to a given Noisy-MAX equation (see Equation 4.31). General form of quality functions is the following:

$$Q : V_n[\mathbb{R}] \times \mathbb{N}^3 \to \mathbb{R} \ , \tag{4.34}$$

where $V_n[\mathbb{R}]$ is a vector of equation coefficients over the field $\mathbb{R}$, and $\mathbb{N}^3$ is a triple: (x, m, M), where

x – (Cnt$(C = +, \mathbf{P})$) – number of records for given a combination of parents, with the child node being activated

m – (Cnt$(\mathbf{P})$) – total number of records for given a combination of parents

M – total number of records

Additionally, we will use the symbol **A** to describe a vector from vector space $V_n[\mathbb{R}]$:

Let us now propose a simple quality function, and show, through an example, how it alters the ordering within the equation set.

Frequency function

Quality function is the following:

$$Q_{FQ}(A, x, m, M) = \frac{m}{M} \ . \tag{4.35}$$

Notice that it does not take parameters A or x into account. This quality function models the idea that we always prefer such combination of parents, that are represented by the biggest fraction of the sample data.

Ordering the initial data from Table 4.3 according to the quality function 4.35 yields Table 4.4. Newly obtained order of equations favors those records that are most represented within the data. This simple idea seems to be quite good at first. After all, higher representation in the sampled data, must go along with better accuracy. However, this is not always the truth in case of the Noisy-MAX equations. In order to explain that, we have to understand the relationship between the JPD of parent nodes, and Equation 4.31.

#	p_1	p_2	p_3	p_L	Cnt($C = +, \mathbf{P}$)	Cnt($\mathbf{P}$)	Q_{FQ}
1	1	0	0	0	300	330	0.33
2	1	0	1	-1	249	263	0.263
3	1	1	0	-1	157	167	0.167
4	1	1	1	-2	134	139	0.139
5	0	0	0	1	1	36	0.036
6	0	0	1	0	4	32	0.032
7	0	1	0	0	2	17	0.017
8	0	1	1	-1	4	16	0.016

Table 4.4: Sample data ordered by quality function Q_{FQ}

If the prior probability of parents favors the distinguished states to occur more often, the resulting sampled data tend to have high number of zero vectors (leak) and the solution vectors. In such a case, Noisy-MAX parameters will be at the top of the ordered system of equations. Naturally, Gauss–Jordan elimination will have a tendency to converge to a simple counting – this is clearly not the case in which the proposed methods can improve anything.

In case when the prior probability of parents favors non-distinguished states, vectors of coefficients at the top would be dominated by vectors with large number of ones. After we analyze the Equation 4.31, we can derive the following:

$$\begin{aligned} P(\neg y|x) = (\neg p_L) \cdot \prod_{i \in I(x)} \frac{\neg p_i'}{\neg p_L} \Rightarrow \\ \Rightarrow P(y|x) = 1 - (1 - p_L) \cdot \prod_{i \in I(x)} \frac{1 - p_i'}{1 - p_L} \, . \end{aligned} \tag{4.36}$$

Notice that the product on the right hand side of Equation 4.36 converges to 1 as the domain of $I(x)$ grows in size. This means that the equations that describe a complicated event (e.g. many possible causes for cancer are present), result in higher probability of the child node being activated. This is, of course, consistent with the intuition, however within the context of learning parameters from data, it makes such records less useful for our purpose. Since we are learning the probability of each combination from a fixed set of records, we cannot express small differences between values near 1 as accurately. This fitness func-

tion does not take any of that information into account – for that reason we can expect it to perform poorly as the number of parents increases.

We will address this problem, after proposing a slightly different quality measure, that is based on exponential penalty.

Confidence function

This quality function is closely related to the confidence interval for proportions – a type of interval estimate often found in statistics:

$$P\left(\frac{x}{m} - u_\alpha\sqrt{\frac{\frac{x}{m}(1-\frac{x}{m})}{m}} < p < \frac{x}{m} + u_\alpha\sqrt{\frac{\frac{x}{m}(1-\frac{x}{m})}{m}}\right) = 1-\alpha\ , \tag{4.37}$$

where

m – total size of the sample – in our case, the total number of records for given combination of parents;

x – number of samples with given trait – in our case, the number of records in which the child node was activated for given combination of parents;

u_α - parameter u_α is not as important for us, since we care about relative differences between breadths of the intervals. We settle on u_α = 1.65, so the confidence interval will describe 95% of the cases.

Quality function uses the width of the interval to estimate how trustworthy a given ratio between x and m is. Narrower intervals suggest better estimate for the parameter.

Quality function is the following:

$$Q_I(A, x, m, M) = \sqrt{\frac{\frac{x}{m}(1-\frac{x}{m})}{m}}\ . \tag{4.38}$$

Table 4.5 presents the ordering of the initial set of equations, using the confidence function above.

#	p_1	p_2	p_3	p_L	Cnt($C = +, \mathbf{P}$)	Cnt($\mathbf{P}$)	Q_I
1	1	0	1	-1	249	263	0.013842995211
2	1	1	1	-2	134	139	0.0157948389112
3	1	0	0	0	300	330	0.0158252414505
4	1	1	0	-1	157	167	0.0183601032631
5	0	0	0	1	1	36	0.0273892582551
6	0	0	1	0	4	32	0.0584633966683
7	0	1	0	0	2	17	0.0781424899006
8	0	1	1	-1	4	16	0.108253175473

Table 4.5: Sample data ordered by quality function Q_I. Records are ordered by the ascending width of the confidence interval.

Coefficient function

At the end of Section 4.4.2, we suggested that the complexity of the equation (the number of non-zero coefficients) plays a major role in the usability of a given parameter. In order to model that, we can propose a quality function that penalizes complex equations.

The quality function is the following:

$$Q_C(A, x, m, M) = m \cdot S^{\sigma} \ , \tag{4.39}$$

where $S = \sum_i^{n-1} A[i]$, is the sum of coefficients for given equation, and σ is the exponent of the penalty component ($\sigma \in \mathbb{R}$, and $\sigma > 1$). Notice that the sum does not take the coefficient of the leak into account ($i \in \{1, \ldots, n-1\}$).

Applying the quality function to the initial set of equations would yield the ordering as presented in Table 4.6.

#	p_1	p_2	p_3	p_L	Cnt$(C = +, \mathbf{P})$	Cnt$(\mathbf{P})$	Q_C
1	1	0	0	0	300	330	116.67
2	1	0	1	-1	249	263	50.61
3	0	0	0	1	1	36	36.0
4	1	1	0	-1	157	167	32.14
5	1	1	1	-2	134	139	17.38
6	0	0	1	0	4	32	11.31
7	0	1	0	0	2	17	6.01
8	0	1	1	-1	4	16	3.08

Table 4.6: Sample data ordered by quality function Q_C, for σ = 1.5

Exponential function

Exponential function aims at modeling the diminishing accuracy of equations with high number of possible causes. Previously, we have introduced the notion that equations with high number of causes present require higher numbers of records in order to maintain the required accuracy. We believe that the additional number of required records grows **exponentially** (as opposed to Q_C, which models that polynomially) with the total number of causes present.

Quality function that captures that idea is as follows:

$$Q_{Exp}(A, x, m, M) = m \cdot \omega^S \ , \tag{4.40}$$

where $S = \sum_i^{n-1} A[i]$, and $0 < \omega < 1$.

We can use the parameter ω to control the penalty component, thus, altering the significance of more complex equations in Gauss–Jordan elimination.

#	p_1	p_2	p_3	p_L	Cnt$(C = +, \mathbf{P})$	Cnt$(\mathbf{P})$	Q_{Exp}
1	1	0	0	0	300	330	165.0
2	1	0	1	-1	249	263	65.75
3	1	1	0	-1	157	167	41.75
4	0	0	0	1	1	36	36.0
5	1	1	1	-2	134	139	17.375
6	0	0	1	0	4	32	16.0
7	0	1	0	0	2	17	8.5
8	0	1	1	-1	4	16	4.0

Table 4.7: Sample data ordered by quality function Q_{Exp}, for ω = 0.5

Exponential function reduces the importance of more complex equations, and in many cases converges to Simple Counting method. However, in cases where the number of records is large enough to overcome the exponential penalty component, we can expect to obtain a better solution for the parameters.

4.4.3 Exploration of zero vectors

Previous section proposes a few quality functions that order the initial set of equations to achieve good initial parameters. In Section 4.2.2, we prove that exploration of possible linear combinations of the initial solution and the zero vectors, can yield other solutions to each of the parameters. In this section we will introduce the methods for gathering other solutions to the parameters.

General framework of exploration

As we had mentioned previously in Section 4.2.2, there are infinitely many solutions for each parameter, provided that we obtain some zero vectors after Gauss–Jordan elimination. Empirical evidence has shown that restriction to linear combination with a fixed set of coefficients, and only to several first zero vectors, yields most efficient solutions. For Z as a number of zero vectors and s_0 as the initial solution to a given parameter, we introduce the

following restriction:

$$s = [1, a_1, a_2, \ldots, a_k] \cdot [s_0, z_1, z_2, \ldots, z_k] \ , \tag{4.41}$$

where $a_i \in \{-\beta, \ldots, -1, 0, 1, \ldots, \beta\}$, and $k < Z$, i.e.,we explore only k first zero vectors – this is especially important, as the number of zero vectors grows exponentially with the number of parents. Additionally, we explore only $2 \cdot \beta + 1$ coefficients for each of the zero vectors.

In order to explore the set of solutions efficiently, we have to introduce yet another quality measure, this time for the linear combination of vectors. Previous quality measures focused on comparing initial equations among each other. This time, we want to compare the fitness of each solution for the parameters $p_1, p_2, \ldots, p_n, p_L$.

#	p_1	p_2	p_3	p_L	P$(C = +\|\mathbf{P})$	Linear Combination
1	1	0	0	0	0.909091	$[1, 0, 0, 0, 0, 0, 0, 0]$
2	0	0	1	0	0.430714	$[-1, 1, 0, 1, 0, 0, 0, 0]$
3	0	1	0	0	0.359614	$[-1, 0, 1, 1, 0, 0, 0, 0]$
4	0	0	0	1	0.027778	$[0, 0, 0, 1, 0, 0, 0, 0]$
5	0	0	0	0	-0.025904	$[1, -1, -1, 0, 1, 0, 0, 0]$
6	0	0	0	0	-0.547013	$[1, -1, 0, -1, 0, 1, 0, 0]$
7	0	0	0	0	-0.377846	$[1, 0, -1, -1, 0, 0, 1, 0]$
8	0	0	0	0	-1.00011	$[2, -1, -1, -1, 0, 0, 0, 1]$

Table 4.8: Sample data ordered by quality function Q_{Exp} (ω = 0.5) and subjected to Gauss–Jordan elimination.

Table 4.8 presents the output of the Gauss–Jordan elimination after the initial ordering of equations using the exponential function (see Section 4.4.2).

General abstract of quality functions for linear combinations is the following:

$$H : V_k[\mathbb{R}] \times \mathbb{R} \to \mathbb{R} \ , \tag{4.42}$$

where $V_k[\mathbb{R}] \times \mathbb{R}$ is a vector of coefficients of linear combination over field $\mathbb{R}$ and the numerical value describing the conditional probability resulting from a given a linear combination.

The vector of coefficients will be denoted by B, and the parameter for conditional probability – λ. In the next section, we will propose an example of fitness quality for linear combination.

Minimal fitness quality

A common way of modeling the global error for a set of random variables is to assume the largest error among them. In order to achieve a similar concept here, we need to assume the minimal fitness among the equations in each linear combination. Some coefficients of the linear combination can be negative, yet, since error is a variance, we can take the absolute value of each of the coefficient. Additionally, the absolute value of some coefficients can be larger than 1 – we want to penalize multiplying the potential error by any constant.

This gives us to the following quality function:

$$H_{Min}(B, \lambda) = \min\left\{ \frac{1}{|B_i|} \cdot Q_{Exp}(< B_i >) \mid i \in \{1..k\} \right\}, \tag{4.43}$$

where $< B_j >$ is the shortened notation for the original set of parameters (A, x, m, M), for the j-th equation in the original system of equations. Fraction $\frac{1}{|B_i|}$ in Equation 4.43 is the weight that penalizes linear combinations with large coefficients.

	p_1	p_2	p_3	p_L	P($C = +$|**P**)	Linear Combination	$H_{Min}(B, \lambda)$
s_1	1	0	0	0	0.909091	$[1, 0, 0, 0, 0, 0, 0, 0]$	165.0
s_2	0	0	1	0	0.430714	$[-1, 1, 0, 1, 0, 0, 0, 0]$	36.0
s_3	0	1	0	0	0.359614	$[-1, 0, 1, 1, 0, 0, 0, 0]$	36.0
s_4	0	0	0	1	0.027778	$[0, 0, 0, 1, 0, 0, 0, 0]$	36.0
z_1	0	0	0	0	-0.025904	$[1, -1, -1, 0, 1, 0, 0, 0]$	17.375
z_2	0	0	0	0	-0.547013	$[1, -1, 0, -1, 0, 1, 0, 0]$	16.0
z_3	0	0	0	0	-0.377846	$[1, 0, -1, -1, 0, 0, 1, 0]$	8.5
z_4	0	0	0	0	-1.00011	$[2, -1, -1, -1, 0, 0, 0, 1]$	4.0

Table 4.9: Previously derived Table 4.8, supplemented with the results of the quality function H_{Min}.

Now that we have means for distinguishing between linear combinations of equations (which is de–facto a quality measure for thus obtained Noisy–MAX parameters), we can

explore different solutions to each parameter using zero vectors. The better the initial ordering of equation, the harder it is to improve any of the original solutions. The information embedded within the zero vectors can be used for two purposes:

1. Finding a better parameter than the original solution s_0, according to some means of comparison

2. Finding a set of parameters close to the original solution s_0 and average them

The Noisy-MAX distribution from which the sample data in Table 4.3 were generated, modeled leak parameter to be equal to 0.01. The initial solution resulted in $p_L = 0.027778$.

If we were to explore some possible linear combinations of s_0 and the zero vectors z_1 – z_4, we would have found solution for p_L equal to $s = [1, 1, 0, 0, 0] \cdot [s_4, z_1, z_2, z_3, z_4] = [1, -1, -1, 1, 1, 0, 0, 0]$. Parameter p_L, when calculated using newly obtained linear combination s, is equal to 0.002593, which is slightly closer than what the original solution s_0 provided. Similarly, taking weighted average of both answers would yield a solution closer than the initial solution:

$$\text{avg}(s_k, s) = \frac{36.0 \cdot 0.027778 + 17.375 \cdot 0.002593}{53.375} = 0.017778\ . \tag{4.44}$$

However, as we have discussed in previous sections, finding such solutions is quite difficult in practice due to large space of exploration and the difficulty of expressing how "good" given solution really is.

5. Implementation and testing

This section will briefly describe tools, programming languages, libraries, and software used for experimentation and testing. Additionally, I will provide details regarding the implementation of our solution, and the experimentation suite I have created for the purpose of this research.

5.1 Tools and software libraries

The core of my solution along with the testing suite was implemented in C++, in order to maintain both speed of execution and flexibility of the implementation. Additionally I supported my implementation efforts with Python (programming language), GeNIe (modeling tool) and libraries: SMILE$^{\text{☺}}$, GSL and SciPy.

5.1.1 GeNIe and SMILE$^{\text{☺}}$

Both GeNIe (Graphical Network Interface) and SMILE$^{\text{☺}}$(Structural Modeling, Inference and Learning Engine) are software packages provided by the Decision Systems Laboratory, School of Information Sciences and the Intelligent Systems Program at the University of Pittsburgh [7]. GeNIe is an easy to use graphical network modeling tool, with large user base (beyond 30 thousand users). Additionally, it is well acclaimed and used by the industry giants such as Intel, Boeing, Oracle or Rockwell Corporation. What is hidden beneath GeNIe, is SMILE$^{\text{☺}}$– an efficient C++ library providing all the functionalities to GeNIe.

GeNIe is a robust tool that allowed us to quickly model test networks and prepare input data for our purpose. In our research, we were using GeNIE and SMILE$^{\text{☺}}$extensively – I generated all of the models using GeNIe, while SMILE$^{\text{☺}}$provided me with efficient handling of the data and generating sample datasets using C++.

5.1.2 GNU Scientific Library

GSL (GNU Scientific Library) is a free and open source software package, providing tools and algorithms for scientific purposes. It is efficiently implemented in C++ programming language and optimized for computation–heavy problems [8]. In our research, we use

repeated experimentation on generated data, in order to draw conclusion using statistical methods. In order to ensure the validity of our approach, we wanted to minimize potential errors that often go along with own implementation of algorithms. For that purpose, we were using GSL's interface for random variable generation using Dirichlet distribution and uniform distribution.

5.1.3 Python and SciPy

Python is a successful and robust programming language that is applied in many areas of computer science [9]. SciPy library provides several extensions to Python, mainly in the domain of scientific computation [10]. Python, along with SciPy is a free software, thus it constituted as a good alternative to commercial software such as MATLAB. We used Python mainly for prototyping, aggregating experimental data, and visualizing the results.

5.2 Implementation details

In this section, I will provide some insight into the general abstract of my implementation efforts.

1. Recognize the matchings between the variables in the datafile and the nodes in the graph
2. Converting records from the datafile into systems of equations (we assume that the samples are from a Noisy-MAX distribution)
3. Solve for the parameters using Gauss–Jordan elimination with different variants as presented in Section 4.4
4. Compare the results to the *gold standard* model that has generated the data, using Euclidian and Hellinger distances.

Points 1 and partially point 4 are provided by the SMILE$^{\text{☺}}$library, as it allows for easy matching of variables and nodes between data files and network structures.

The abstract prototype for all of our methods is as follows:

$$F : V \times E \times D \to \mathbb{T} \, . \tag{5.1}$$

We treated the structure of the network (graph (V, E)) as one of the **input** parameters, along with the data source (denoted as D). The **output** of our solutions are the sets of conditional probability distributions (family $\mathbb{T}$).

Testing suite uses the following abstract of comparison methods:

$$\text{Dist} : \mathbb{T} \times \mathbb{T}_G \rightarrow \mathbb{R} \,, \tag{5.2}$$

where $\mathbb{T}$ are the results obtained by us or any of the reference methods (e.g., Simple Counting) and $\mathbb{T}_G$ - distribution from which a given datafile D was generated (gold standard). The output is a real number describing distance between probability distributions $\mathbb{T}$ and $\mathbb{T}_G$.

Implementation efforts were a quite straightforward task in itself, however due to the experimental nature of our endeavour, we cared about flexibility more than efficiency. Some of the theoretical models for solving Noisy-MAX parameters, described in Section 4.4, initially arose from the experiments, and trial and error approach.

5.2.1 Handling missing or incorrect parameters

As the proportion of records per parameter gets smaller, some parameters are either represented by singular records or none at all. Some methods handle this better than others, however some reference methods (e.g., simple counting) do not.

Missing parameters for Simple Counting

Simple Counting sweeps through the data file looking for the bare Noisy-MAX parameters. The cases when some parameters cannot be accurately computed from the data file, can be of 3 possible types:

1. Parameter is equal to 0.0 – it happens when for every instance of given combination of parents, the child node is in its non–distinguished state

2. Parameter is equal to 1.0 – Similar scenario as above: for every instance of given combination of parents, the child node was activated

3. Parameter is undefined – No occurrences of a given Noisy-MAX parameter were found in a data file.

This could be resolved using two approaches:

- Take the fixed value: For a fixed value of ϵ, we handle scenarios 1, 2 and 3 by assuming the missing parameters as ϵ, $1 - \epsilon$ and 0.5 respectively.
- Take the closest relative value: For x as the number of records that activated the child, and m as the total number of records for given combination of parents, we handle scenarios 1, 2 and 3 by values $\frac{x+1}{m}$, $\frac{x}{m+1}$ and 0.5.

Second approach proved to achieve overall better results that the fixed value of ϵ. For that reason I chose it as the default method for handling errors in Simple Counting.

Incorrect parameters for Noisy–MAX

Missing parameters for Noisy–MAX is not as much of a problem for Gauss–Jordan based methods as long as the system of equations contains enough linearly independent vectors to solve for every unknown. However, they are not immune to errors originating from insufficiently sampled data.

When working with Gauss–Jordan based methods, we may encounter solutions that are simply incorrect, e.g., probabilities $p_i < 0$. This results from the problem of leak being poorly sampled, thus, biased with high error (see Section 4.3). If we look at Henrion's definition of parameters obtained from data (Equation 4.31), we can notice that the product on the left hand side of the equation contains leak in the denominator. For parameters $\neg p_L$, disproportionately small to the parameter $\neg p'_i$, we can obtain a product that is greater than 1. This results in the conditional probability being less than 0.

Error handling in such cases is far more difficult to handle. Similarly to the previous solution, we can agree on a fixed value of ϵ and handle the errors accordingly:

- In case when the obtained parameter is incorrect, assume the ϵ.

Unfortunately we do not have yet any good candidate for relative value counterpart as it is used for Simple Counting.

5.3 Testing suite and experiments

This sections will present the experiments by which I compared the efficiency of the methods for learning the Noisy-MAX parameters.

5.3.1 Networks and data sets

The testing was done on generated models, defined using GeNIe [7]. Without loss of generality, we modeled each network as consisting of 1 child node and k parents. Number of parents varied from 2 up to 12.

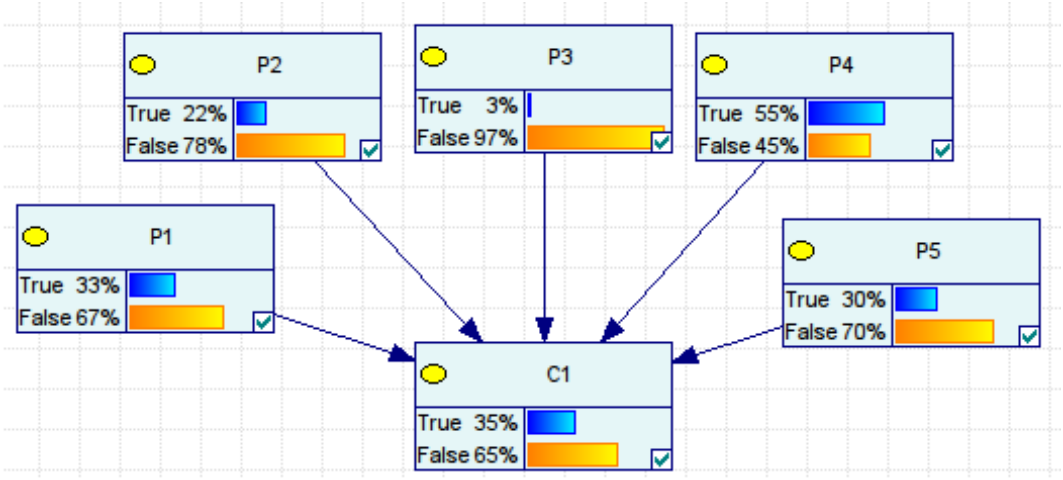

Figure 5.1: Noisy-OR model with 5 parents.

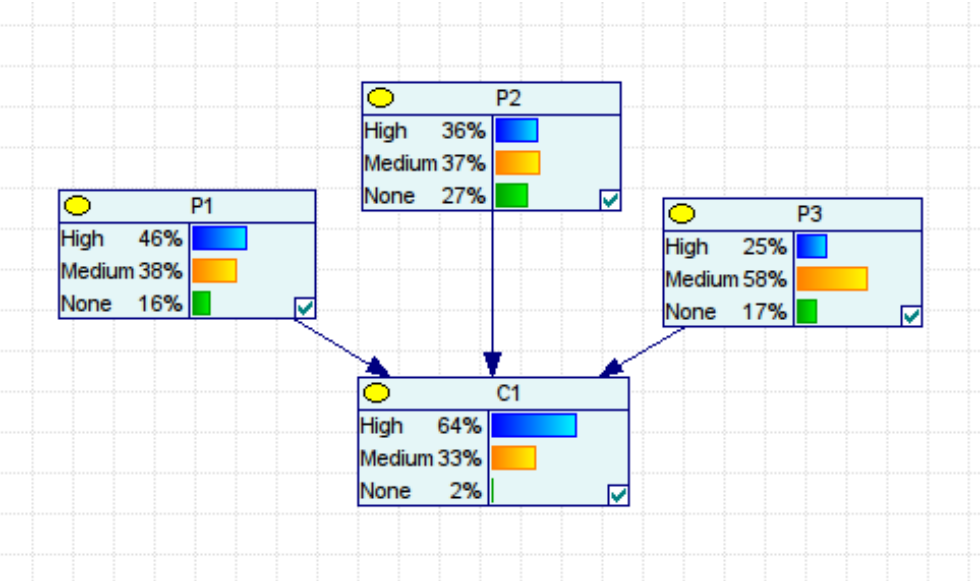

Figure 5.2: Noisy-MAX model with 3 parents.

Figures 5.1 and 5.2 present screen shots of models *OR_5.xdsl* and *MAX_3_3.xdsl* respectively.

As for the non-binary models, we tested Noisy-MAX networks for sizes from 2 up to 5 parents, with number of outcomes for each node varying from 3 to 5.

Dataset sizes changed semi–exponentially from 100 records up to 50.000.

Randomization of gold standards

When modeling the networks, we did not use constant parameters for the CPT's. The parameters are randomized using the uniform and the Dirichlet distributions during the experiments, in order to assure statistical significance of the results. Each experiment started with randomizing the network parameters using uniform distribution. Tests that required the noising of the parameters were once again randomized, this time from the Dirichlet distribution. We randomized prior probability distributions for each of the parents, as well as the Noisy-MAX parameters for the child node.

5.3.2 Types of experiments

In the experiments I have tested the following variables:

1. **M** - Methods for learning Noisy-MAX distribution
2. **D** - Size of the data sample
3. **P** - Number of parents
4. **K** - Parameter describing noise, introduced using Dirichlet distribution

I divided the experiments into two classes:

1. Investigating the impact of size of the datafile and the number of parents, on the quality of the Noisy-MAX parameter learning. This class of experiments tests the variables **M**, **D** and **P** for various settings.
2. Examining the behavior of learning methods when the original network is not a Noisy–MAX. This class of experiments tests the variables **M**, **D**, **P** and **K**, although it is done in a less extensive manner, due to the introduction of the additional variable **K**.

Introducing noise by means of the Dirichlet distribution

Dirichlet distribution is often thought of as a family of multivariate probability distributions:

$$\text{Dir}(x_1, \ldots, x_{K-1}; \alpha_1, \ldots, \alpha_K) = \frac{1}{\text{B}(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1} , \tag{5.3}$$

where B is a multinomial Beta function, α_1 – α_K are the numerical parameters often called the *concentration parameters*, and x_1 – x_{K-1} are the random variables of the distribution.

We use the definition above in order to introduce a distortion to the ideal parameters of Noisy-MAX. Dirichlet distribution was chosen, because it allows for distorting the CPT of the child node. We do this, by introducing a noising parameter κ. We subject the original distribution $\theta = [\theta_1, \theta_2, \ldots, \theta_{n-1}]$ in to distortion by obtaining a random variable $\theta' = [\theta'_1, \theta'_2, \ldots, \theta'_{n-1}]$ from the Dirichlet distribution:

$$f := Dir(x_1, x_2, \ldots, x_{n-1}; \kappa \cdot \theta_1, \kappa \cdot \theta_2, \ldots \kappa \cdot \theta_{n-1}) , \tag{5.4}$$

$$\theta' = \text{Rnd}(f) , \tag{5.5}$$

where $\text{Rnd}(f)$ is a function returning a random variable from a probability density function f.

For $\kappa >= 10,000$, little to no noise is introduced to the original distribution θ. As κ decreases to $\kappa < 10$, original distribution is distorted. Our noising experiments simply assume that **K**= κ, and obtain a random variable using the method described above.

5.3.3 Euclidian and Hellinger metrics

In order to compare two methods we require a metric function. We have settled on the Euclidian distance and the Hellinger distance as the means of comparison. Let us assume that $\mathbb{P}$ is a CPT of the size $I \times J$, and $\mathbb{P}^i_j$ is the i-th row and the j-th column of $\mathbb{P}$.

Additionally, each distance will be defined for two variants: MAX and AVG. Since both distances are defined for a singular distribution, we need to aggregate a set of distances between distributions into a single value. For that purpose we either take the average distance, or the maximum.

Euclidian distance

$$D_E(\mathbb{P}, \mathbb{Q}, j) = \sqrt{\sum_{i=1}^{I} (\mathbb{P}^i_j - \mathbb{Q}^i_j)} \tag{5.6}$$

$$D_{E[MAX]}(\mathbb{P}, \mathbb{Q}) = \max\{D_E(\mathbb{P}, \mathbb{Q}, j) \mid j \in J\} \tag{5.7}$$

$$D_{E[AVG]}(\mathbb{P}, \mathbb{Q}) = \text{avg}\{D_E(\mathbb{P}, \mathbb{Q}, j) \mid j \in J\} \tag{5.8}$$

Hellinger distance

$$D_H(\mathbb{P}, \mathbb{Q}, j) = \frac{1}{\sqrt{2}} \cdot \sqrt{\sum_{i=1}^{I} \left(\sqrt{\mathbb{P}^i_j} - \sqrt{\mathbb{Q}^i_j}\right)^2} \tag{5.9}$$

$$D_{H[MAX]}(\mathbb{P}, \mathbb{Q}) = \max\{D_E(\mathbb{P}, \mathbb{Q}, j) \mid j \in J\} \tag{5.10}$$

$$D_{H[AVG]}(\mathbb{P}, \mathbb{Q}) = \text{avg}\{D_E(\mathbb{P}, \mathbb{Q}, j) \mid j \in J\} \tag{5.11}$$

5.3.4 Visualisation of the results

Main variable in our research is the variable **M**, and their response to varying variables **D**, **P** and **K**. In order to obtain reliable results, we repeat the experiment for given state of variables 1,000 times – this permits us to examine the results statistically. Box plot is an excellent form of visualisation for this kind of experiments.

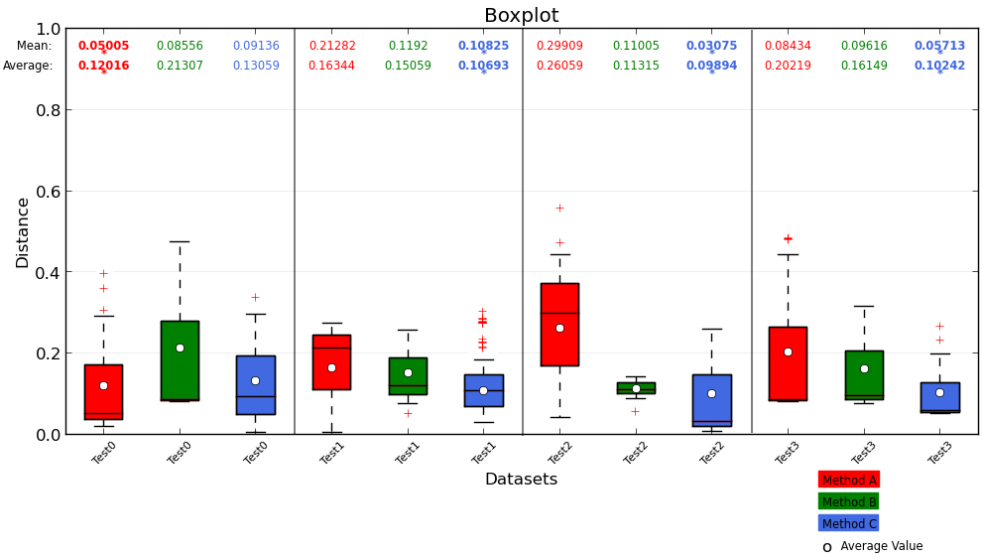

Figure 5.3: Example of a box plot.

We will describe each characteristic of the box plot by referring to the Figure 5.3.

1. Colored box describes the interquartile range (IQR),i.e., 50% observations that fall between Q_1 and Q_3
2. Whiskers of the box plot are the lower and upper boundary between which 99.3% of all observations fall – interval $Q_1 - 1.5 \cdot \text{IQR}, Q_3 + 1.5 \cdot \text{IQR}$
3. Outliers are marked with red "plus" sign (+)
4. Horizontal bar inside the IQR marks the mean, while the white dot marks the average
5. Mean and average are also presented as a number at the top of the plot (smallest values among the methods are boldfaced)

A single plot contains the comparison of several methods, distinguished by different colors.

Comparison above is done per each dataset (either varying **D** or **K**), thus, they are divided by vertical lines into groups.

6. Experimental results

This chapter presents the results of the statistical experiments, based on which we will draw conclusions and prove or disprove the conjectures proposed in the previous, theoretical chapters. Each experiment is described in a separate section, along with the observations and conclusions.

6.1 Notation

Before we present the results, let us provide a guide for the variables that we will test. Below, we describe the domains of tested variables.

Domain of M

1. Q_{FQ} – Gauss–Jordan elimination, ordered by the Frequency function (see Section 4.4.2)
2. Q_I – Gauss–Jordan elimination, ordered by the Confidence function (see Section 4.4.2)
3. $Q_C[\sigma = \sigma_0]$ – Gauss–Jordan elimination, ordered by the Coefficient function (see Section 4.4.2), with the parameter σ equal to σ_0
4. $Q_{Exp}[\omega = \omega_0]$ – Gauss–Jordan elimination, ordered by the Coefficient function (see Section 4.4.2), with the parameter ω equal to ω_0
5. Simple – Simple Counting method.
6. SMILE – Noisy–MAX fitting, as it is implemented in SMILE$^{\text{☺}}$.

Domain of D

1. OR$_M$ – A datafile generated from the Noisy–OR network, consisting of M records.
2. MAX$_M^J$ – A datafile generated from the Noisy–MAX network, consisting of M records, where each variable (including the child) has J possible outcomes

Domain of P

Variable **P** is an integer describing the number of parents. Lower boundary of the domain is 2 – we require at least two parents to have a meaningful Noisy-MAX distribution. Upper boundary, theoretically, does not exists, although computing parameters for $\mathbf{P} > 15$ is cumbersome, and in many cases does not yield any more insight into the results anyway. In my experiments, I used $2 \leq \mathbf{P} \leq 12$.

Domain of K

Variable **K** (noising parameter) is a real positive number. **K** will usually change, semi–logarithmically, from $\mathbf{K}= 10.000$ (little to no noise), up to $\mathbf{K}= 2.0$ (large amount of noise).

Additionally, some experiments may propose some custom variables (such as parameters σ or ω). We do not introduce the abstract notation for such variables – all special cases will be mentioned for each of the experiments individually.

6.2 Method accuracy

This class of experiments will test the accuracy with which the methods learn the Noisy-MAX parameters. Variables altered in order to provide a more difficult task will be **D** – data size, and **P** – the number of parents.

6.2.1 Initial Test

The following experiment will test the change in accuracy of all the methods, for the increasing number of records and parents. Our possible space of experimentation is vast. This is the initial test that involves all of the methods, in order to distinguish the better performing ones, and focus on more in–depth testing for them in separation.

Tested variables

- **M** = $\{Q_{FQ}, Q_I, Q_C[\sigma = 3], Q_{Exp}[\omega = 0.3],$ Simple Counting$\}$
- **D** = $\{\text{OR}_{50}, \text{OR}_{200}, \text{OR}_{800}, \text{OR}_{3200}\}$

- **P** = $\{2, 4, 8\}$
- We will use Hellinger[AVG] distance for the comparison.

Results of the experiment can be seen on Figure 6.1.

Observations

1. Among the Gauss–Jordan based solutions, method Q_{FQ} tends to have a higher variance than methods $Q_C[\sigma = 3]$ and $Q_{Exp}[\omega = 0.3]$.

2. Method Q_I performs relatively poorly, when compared to the remaining Gaussian methods – even the Q_{FQ} variant, which was diagnosed as a mediocre quality function, early in the development of this research. This is quite surprising, and contrary to our expectations. The concept of using the width of the confidence interval was very appealing to us due to its sound statistical foundation. If we focus on the plot at the very bottom of Figure 6.1 we can notice that the accuracy of Q_I diminishes (relatively to the remaining methods) as variable S increases – this is especially interesting, since it means that the method does not utilize the extra amount of information at all.

3. Methods Q_C and Q_{Exp} achieve similar accuracy, although Q_{Exp} tends to result in slightly lower mean and averate.

4. If we look at the mean and the average, it seems that the best performing method for **P** = 4 (and partially for **P**=2 and **P**=8 as well) is $Q_{Exp}[\omega = 0.3]$, although improvement over the Simple Counting method is not particularly large.

5. In the combination of large number of parents (**P**=8) and small amount of data (**D** = OR_{50} and OR_{200}), Simple Counting performs significantly better. This is particularly interesting, as it is contrary to our original expectation that we can learn the Noisy–MAX parameters better using Gauss–Jordan elimination, especially when the number of records is small. One of the reasons behind that could be the lack of good error handling mechanism for Gauss–Jordan elimination (see Section 5.2.1). Simple Counting method can settle the cases of missing or incorrect parameters relatively easy; in case of Gauss–Jordan elimination, we have to rely on a fixed ϵ value. However, as

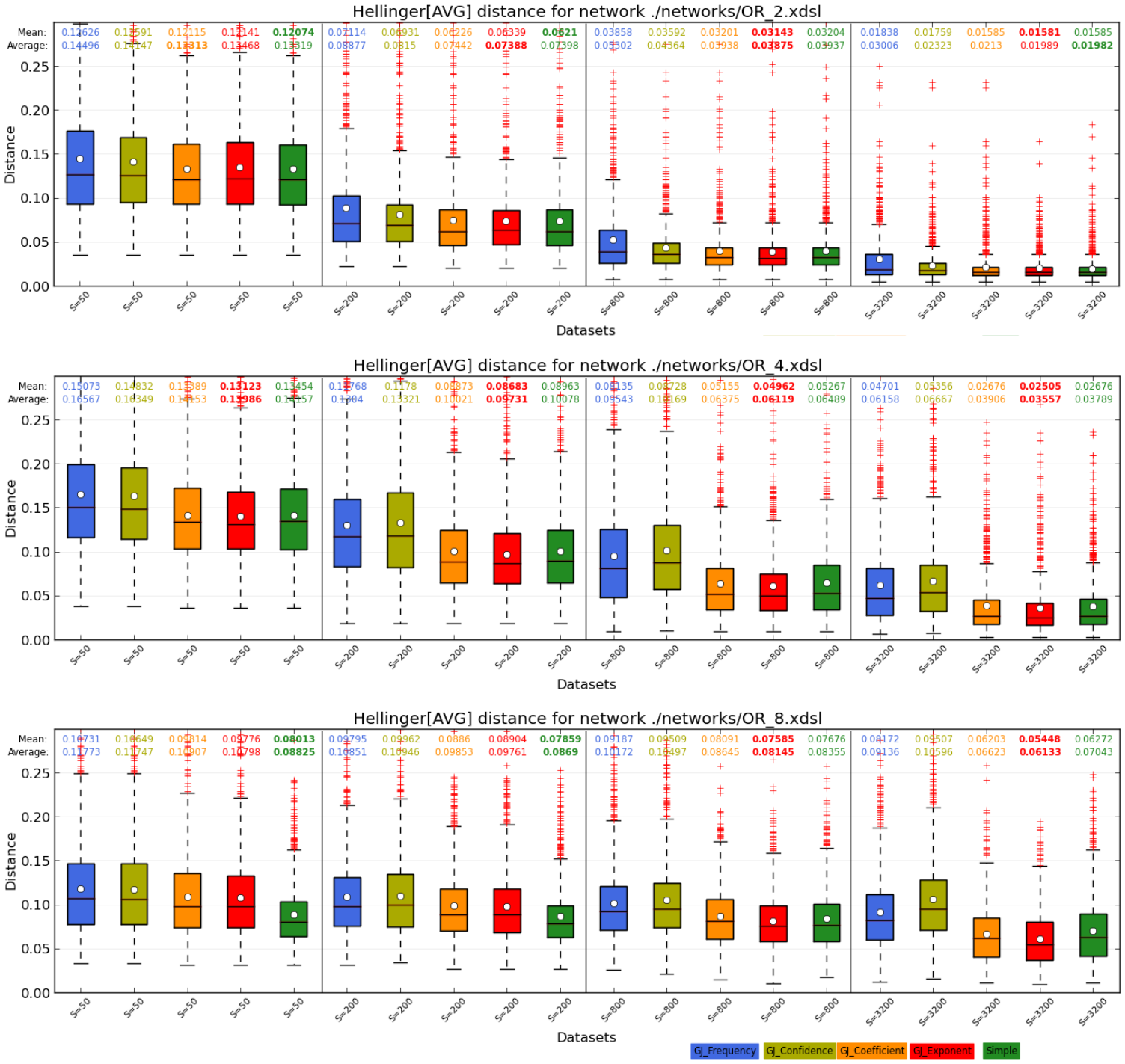

Figure 6.1: Initial test for all of the methods.

the variable D progresses towards larger datasets (OR_{800} and OR_{3200}), Simple Counting method is no longer the best performing candidate, and is replaced by method $Q_{Exp}[\omega = 0.3]$ again.

Conclusions

1. Let us compare the differences between methods $Q_C[\sigma = 3]$ and $Q_{Exp}[\omega = 0.3]$. It seems that exponential penalty does seem to model the diminishing accuracy of more complex equations slightly better. This is at least true for the assumed parameters σ and ω. Testing this relationship for varying values of σ and ω may bring us to a better understanding of the mechanisms behind the increasing error in complex equations.

2. Contrary to our initial suggestion, Gauss–Jordan based methods may actually work better for the larger number of records, that is, use more information from the data in order to get more accurate results.

As we can see, some of the proposed Gauss–Jordan based methods do not improve the accuracy over the Simple Counting significantly. Our guess is that it comes from the fact described in the last paragraph of Section 4.3.1 – poor learning of the leak parameter is causing the product of Equation 4.31 to introduce error. However, Gauss–Jordan method $Q_{Exp}[\omega = 0.3]$ seems to perform quite well, and in most cases is better than Simple Counting approach.

6.2.2 Testing the Coefficients Quality function

Previous experiment had shown that Q_C and Q_{Exp} perform the best among the Gaussian methods. The following experiment will test the impact of the parameter σ on the learning accuracy of the Gauss–Jordan based method Q_C. In order to find the best possible variant of Q_C.

Tested variables

- **M** = $\{Q_C[\sigma = \sigma_k]\}$
- σ_k = $\{1.0, 1.5, 2.0, 3.0\}$

- **D** = $\{\text{OR}_{50}, \text{OR}_{200}, \text{OR}_{800}, \text{OR}_{3200}\}$
- **P** = $\{2, 4, 8\}$
- We will use Hellinger[AVG] distance for the comparison.

Results of the experiment are shown in Figure 6.2.

Observation

1. It seems that parameters σ has a small effect on the learning accuracy. Parameter $\sigma = 1$ is almost as good as penalizing the complex equation using $\sigma = 3$. The minuscule differences tend to favor parameter $\sigma = 3$, especially for **D**=OR_{50}.

Conclusion

1. Since the role of parameter σ is small, we can safely assume the value $\sigma = 3$ as the best.

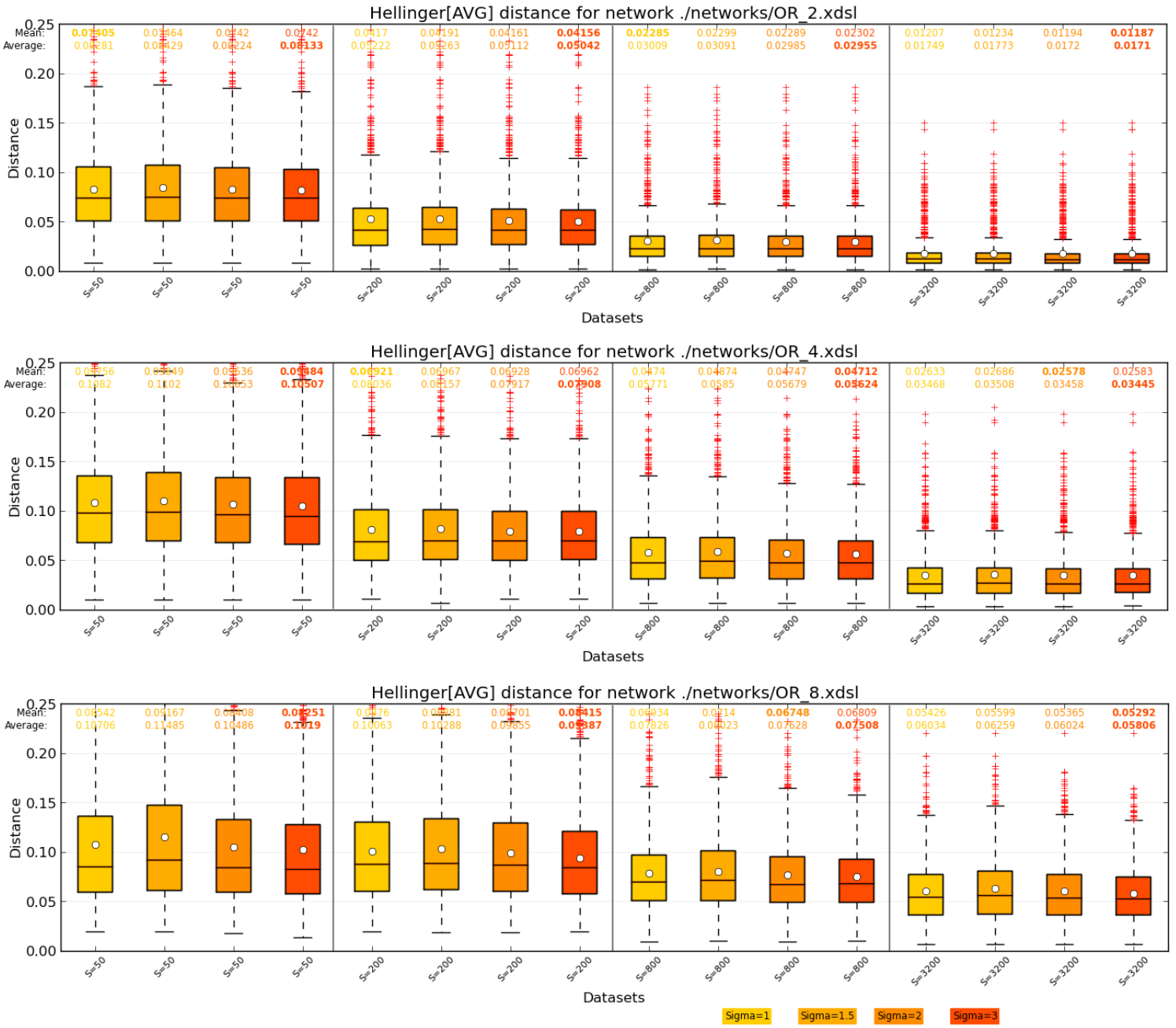

Figure 6.2: Results of the varying parameter σ on the method Q_C.

6.2.3 Testing the Exponential Quality function

The following experiment will test the impact of the parameter ω on the accuracy of the Gauss–Jordan based method Q_{Exp}. We will change the value of ω in a following progression: $0.2, 0.3, 0.4, 0.7$. The last parameter is a control value, designed to show that the higher base of the exponent does not penalize the complex equations enough.

Tested variables

- **M** = $\{Q_{Exp}[\omega = \omega_k]\}$
- ω_k = $\{0.2, 0.3, 0.4, 0.7\}$
- **D** = $\{\text{OR}_{50}, \text{OR}_{200}, \text{OR}_{800}, \text{OR}_{3200}\}$
- **P** = $\{2, 4, 8\}$
- We will use Hellinger[AVG] metric for the comparison.

Results of the experiment are shown in Figure 6.3.

Observation

1. It is especially evident for the variable **P**=8 that the parameter $\omega = \{0.2, 0.3\}$ tends to narrow down the variance of the Q_{Exp} better than higher values of ω. Similarly to the previous experiment, the relative difference between parameters is small (except for the $\omega = 0.7$, which was a control value to test the base of the exponent that is larger than 0.5).

Conclusion

1. Parameter $\omega = \{0.2, 0.3\}$ seems like a good base for the exponential penalty component. We assume $\omega = 0.3$ as the default parameter.

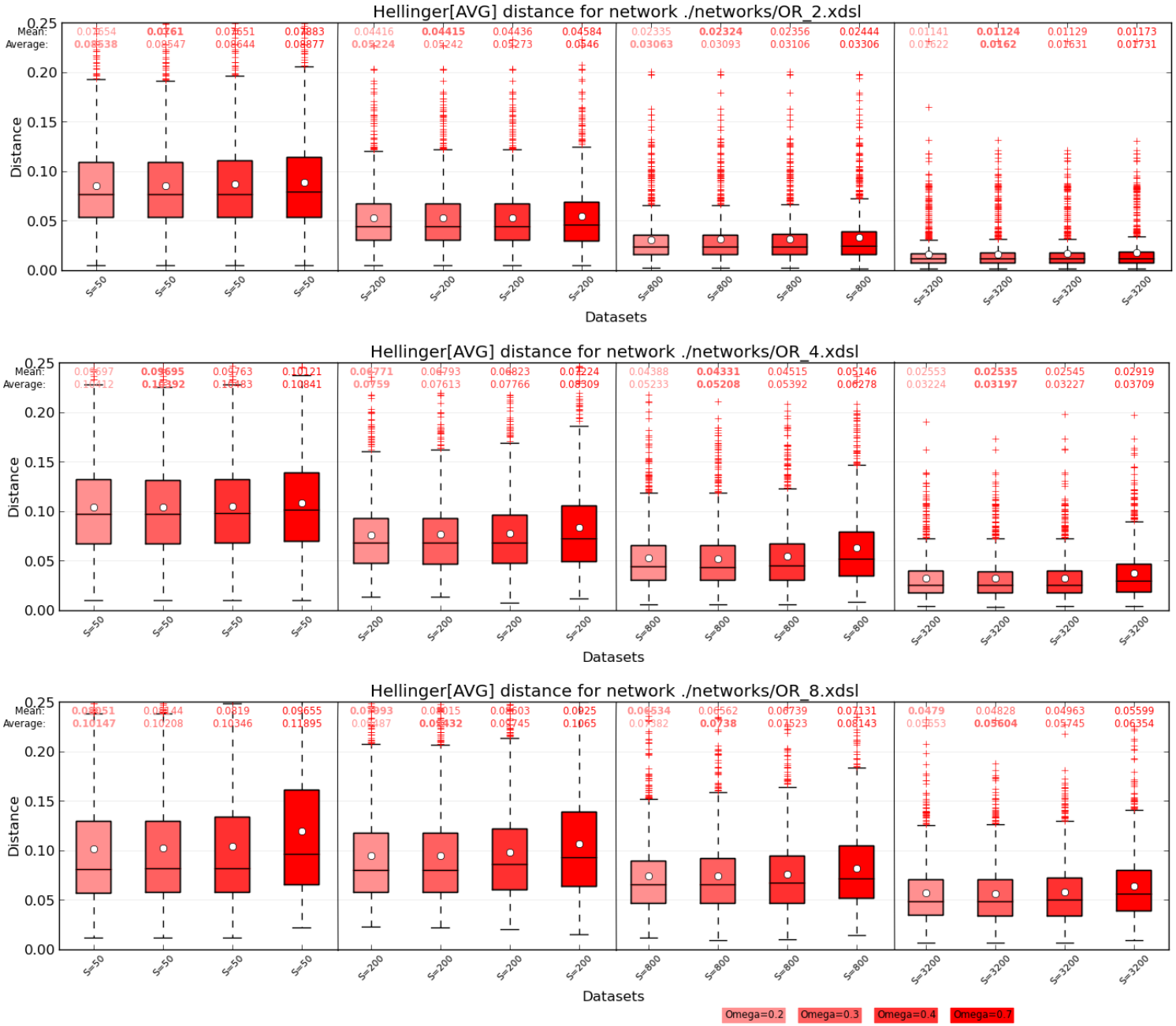

Figure 6.3: Results of the varying parameter ω on the method Q_{Exp}.

6.2.4 Selecting the champion method

The following experiment will compare the methods $Q_C[\sigma = 3]$ and $Q_{Exp}[\omega = 0.3]$.

- **M** = $\{Q_C[\sigma = 3], Q_{Exp}[\omega = 0.3]\}$
- **D** = $\{\text{OR}_{50}, \text{OR}_{200}, \text{OR}_{800}, \text{OR}_{3200}, \text{OR}_{12800}\}$
- **P** = $\{2, 8\}$
- We will use distances Hellinger[AVG] and Hellinger[MAX] for the comparison.

Results of the experiment are shown in Figure 6.4.

Observation

As we can see in almost every case, method $Q_{Exp}[\omega = 0.3]$ dominates the Coefficient–based solution.

Conclusion

It seems that the complex equations are indeed less useful, as the number of present causes increases. The rate at which it happens seems to be exponential to the number of non–zero coefficients of a given equation. This is especially useful knowledge when designing another Noisy-MAX learning method, more sophisticated than Simple Counting.

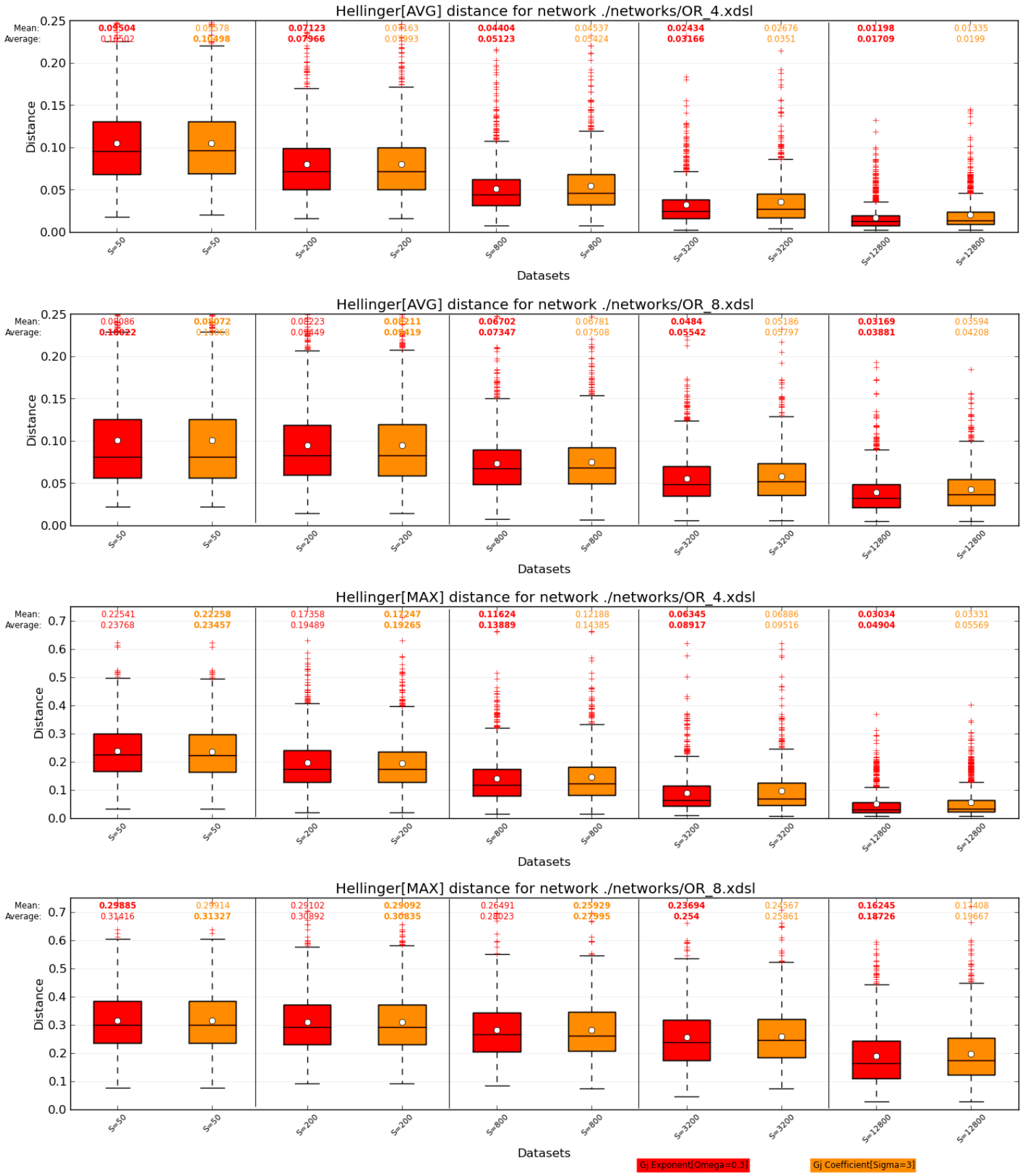

Figure 6.4: Comparison of methods Q_C and Q_{Exp} for the best variants of σ and ω.

6.2.5 The best Gauss–Jordan based method compared with Simple Counting

The following experiment will test the change in accuracy of the best Gauss–Jordan based method, as concluded from the previous test, and the Simple Counting method.

Tested variables

- **M** = $\{Q_{Exp}[\omega = 0.3], \text{Simple Counting}\}$
- **D** = $\{\text{OR}_{100}, \text{OR}_{1000}, \text{OR}_{5000}, \text{OR}_{20000}, \text{OR}_{50000}\}$
- **P** = $\{2, 4, 6, 8, 12\}$
- We will use the Hellinger[AVG], Hellinger[MAX], Euclidian[AVG] and Euclidian[MAX] distances.

Results of the experiment can be seen on Figure 6.7.

Observations

1. Gauss–Jordan based method $Q_{Exp}[\omega = 0.3]$ indeed performs better than Simple Counting, but only after a certain number of records has been provided. We can draw the following observations from the Figure 6.7:
   (a) For **P** = 2, $Q_{Exp}[\omega = 0.3]$ performs better than Simple Counting for every state of **D**.
   (b) For **P** = 4, $Q_{Exp}[\omega = 0.3]$ performs better than Simple Counting for **D** being somewhere in-between 100 and $1,000$ records.
   (c) Similarily, for **P** = 6, breaking point of **D** is somewhere near $1,000$ records (and so on).
2. The turnover point is shifting towards the larger number of records, as the number of parents increases.

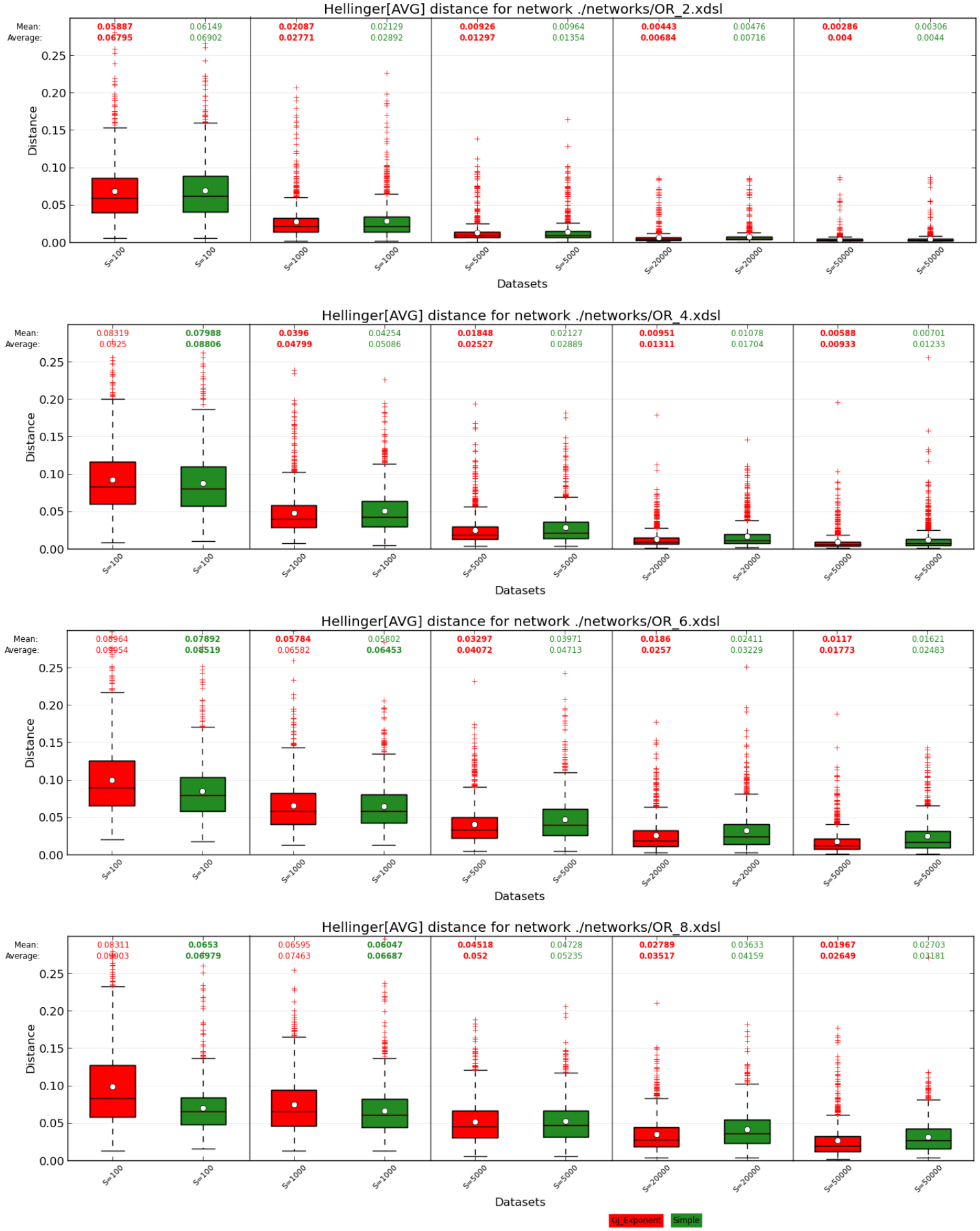

Figure 6.5: Comparison of $Q_{Exp}[\sigma = 0.3]$ and the Simple Counting method.

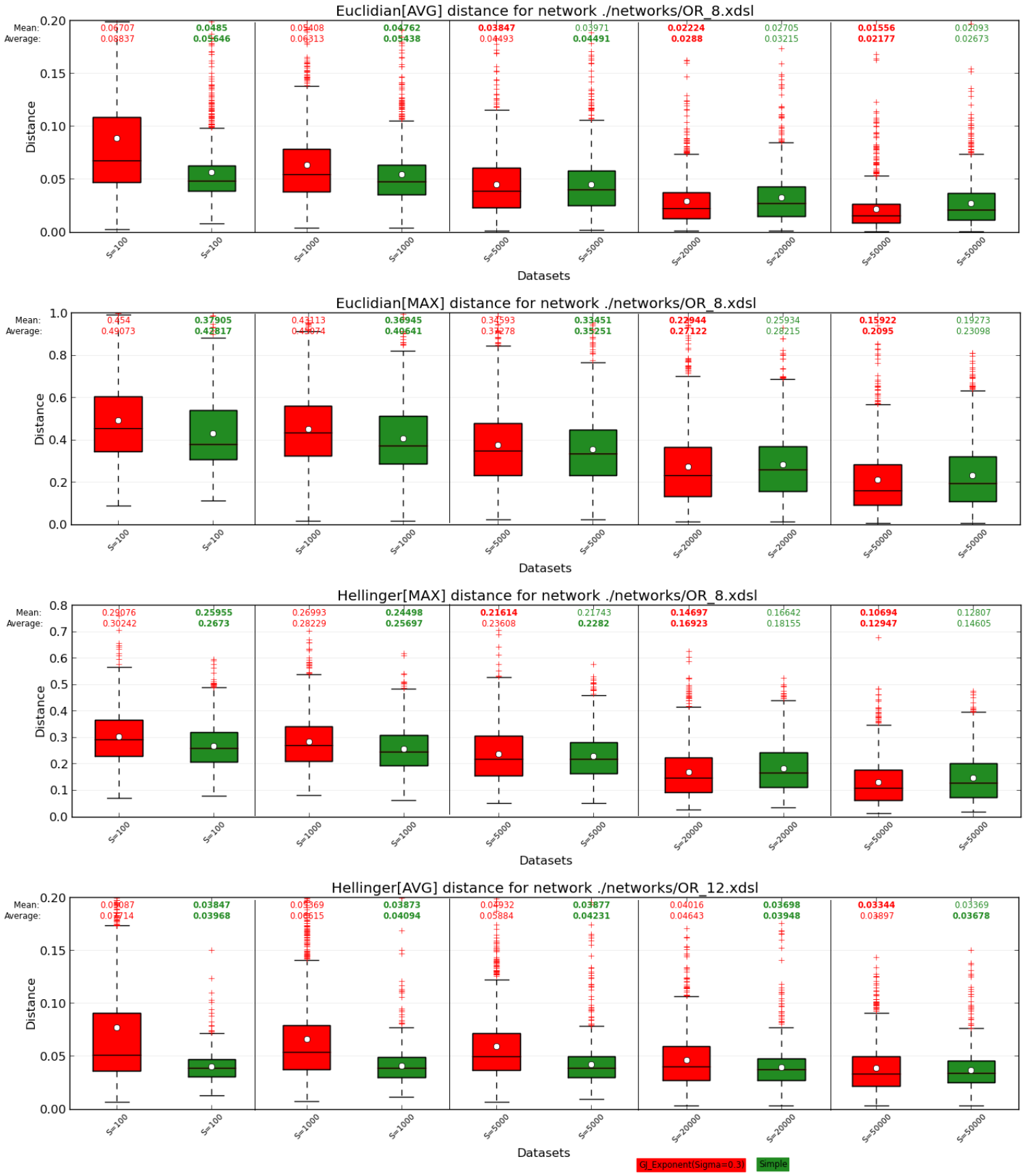

Figure 6.6: Comparison of $Q_{Exp}[\sigma = 0.3]$ and the Simple Counting method for 8 and 12 parents, using different distances.

Conclusions

1. Observations above may suggest that the solution Q_{Exp} does not necessarily work better in cases of small number of records. However, due to using larger amount of information from data, it is able to find better solutions for each of the parameters when given a large enough dataset.

2. In the last plot on Figure 6.6 we can notice that the turnover point is larger than $50,000$. We can expect to find a large enough dataset, for which the Q_{Exp} will perform better than Simple Counting, yet the relatively small number of parents (12) requiring quite a large number of records $(50,000)$ are already large demands.

6.2.6 Comparison with SMILE

The following experiment will test the change in accuracy of the best Gauss–Jordan based method from the previous test, the Simple Counting, and the solution implemented in SMILE$^{\text{☺}}$.

Tested variables

- **M** = $\{Q_{Exp}[\omega = 0.3], \text{Simple Counting}\}$
- **D** = $\{\text{OR}_{100}, \text{OR}_{1000}, \text{OR}_{5000}, \text{OR}_{20000}, \text{OR}_{50000}\}$
- **P** = $\{2, 3, 4\}$
- We will use the Hellinger[AVG] and the Hellinger[MAX] distances for the comparison.

Results of the experiment can be seen on Figure 6.7.

Observations

1. For the testing variables **D**$= 100$ and **P**$= 2$, SMILE method dominates both the Simple Counting and the Gauss–Jordan based method.

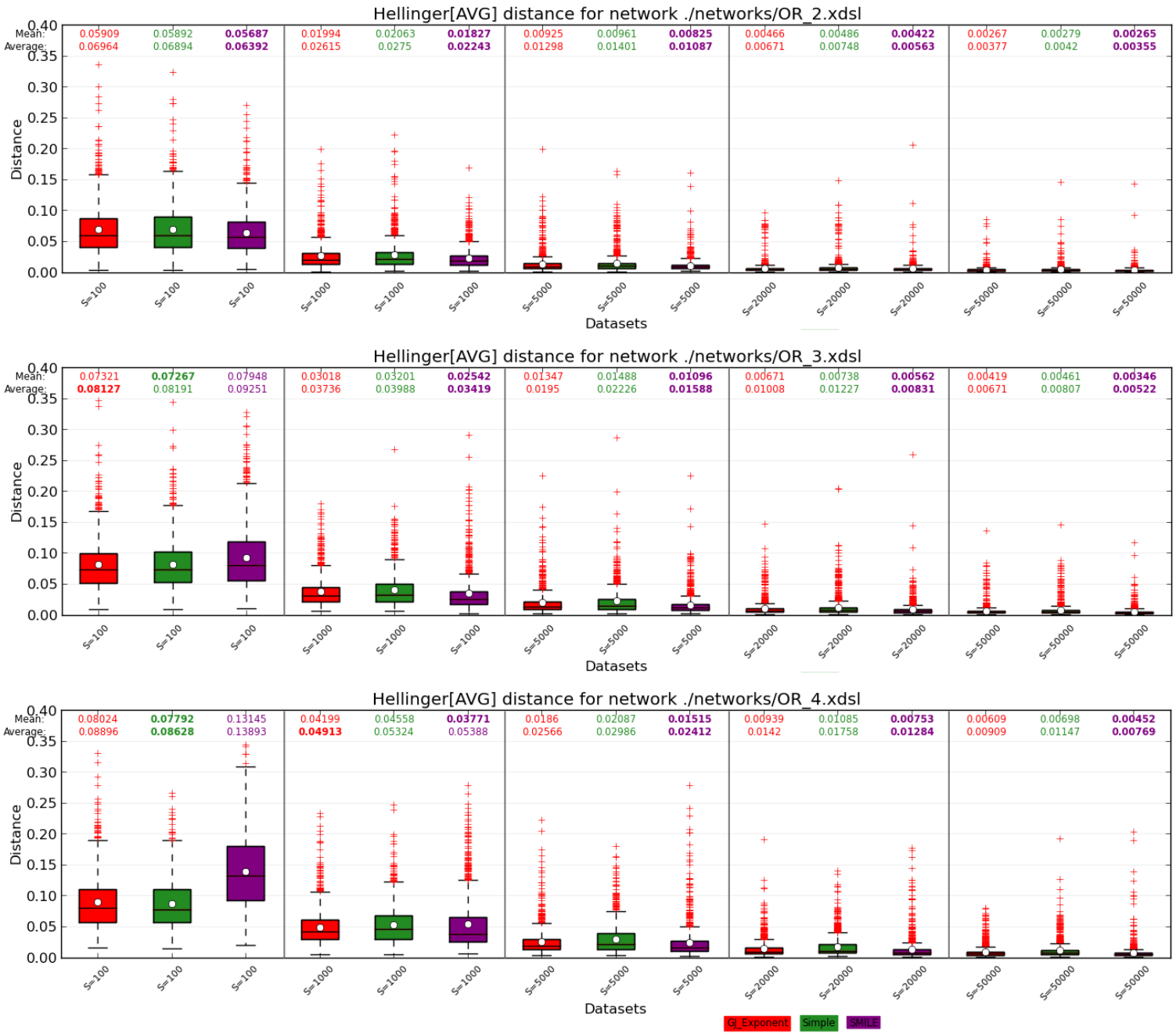

Figure 6.7: Comparison of $Q_{Exp}[\sigma = 0.3]$, Simple Counting and the SMILE – Hellinger[AVG].

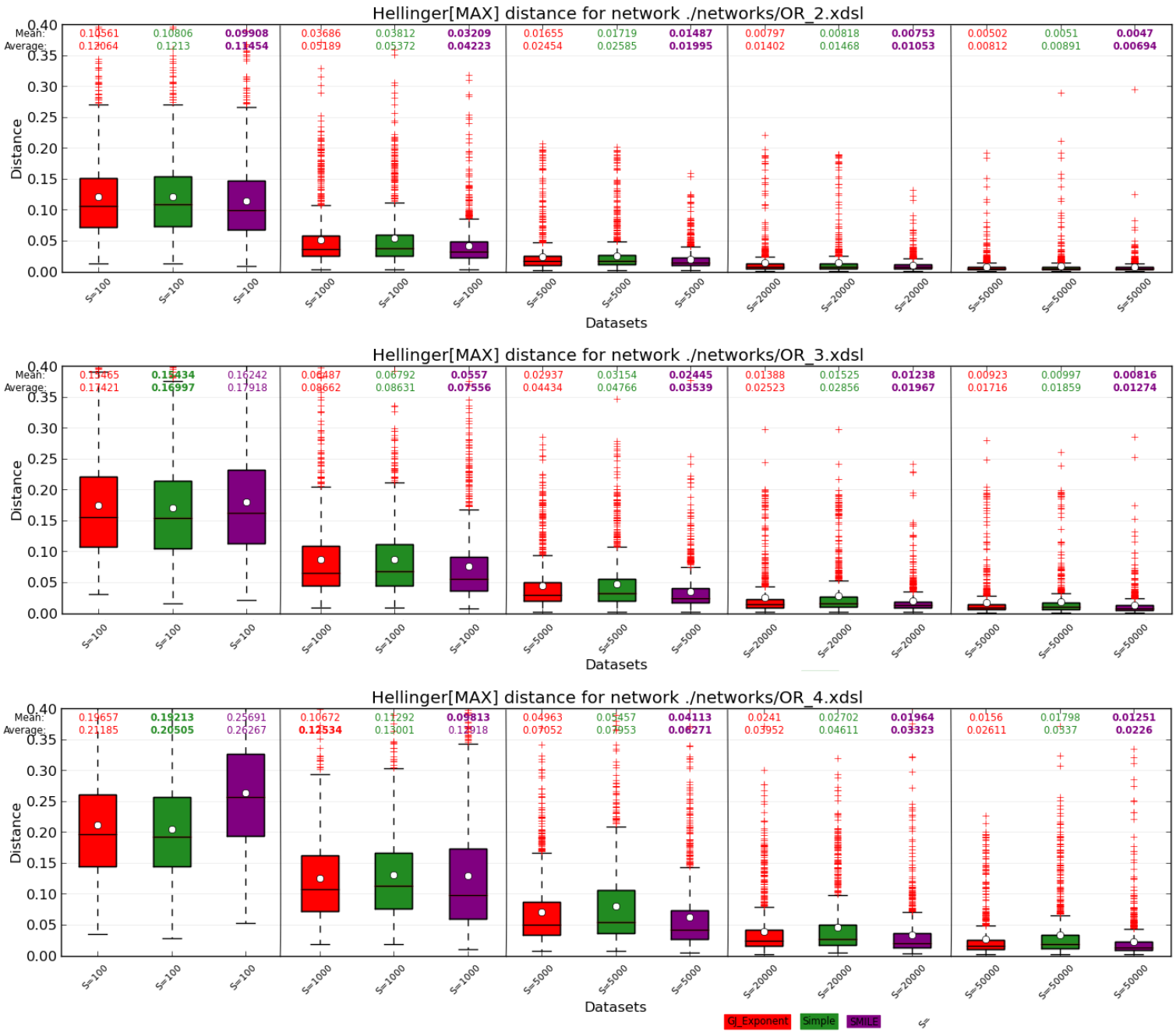

Figure 6.8: Comparison of $Q_{Exp}[\sigma = 0.3]$, Simple Counting and the SMILE – Hellinger[MAX].

2. As **P** increases, and **D** stays the same, SMILE begins to fit Noisy–MAX distributions worse than the remaining methods.

Conclusion

SMILE provides a good method for learning in general case, however, in case of low number of records, Simple Counting may yield better results with relatively small effort in terms of both, implementation and computational complexity.

6.2.7 Comparison for non–binary models

The following experiment will test the change in the accuracy of the Gauss–Jordan based methods, and the Simple Counting when appied to non binary cases of Noisy–MAX.

Tested variables

- **M** = $\{Q_{FQ}, Q_I, Q_C[\sigma = 3], Q_{Exp}[\omega = 0.3], \text{Simple Counting}\}$
- **D** = $\{\text{MAX}^J_{50}, \text{MAX}^J_{200}, \text{MAX}^J_{800}, \text{MAX}^J_{3200}\}$
- **J** = $\{3, 4, 5\}$
- **P** = $\{3, 4, 5\}$
- We will use the Hellinger[AVG] distance for the comparison.

We introduce a new variable for this test: J – number of outcomes of each of the nodes. In this tests J will change accordingly to P.

Results of the experiment can be seen on Figure 6.9.

Observations

1. It seems that the the Simple Counting methods dominates others in terms of accuracy for non binary models.
2. The relative difference among the Gauss–Jordan based methods is Similar to previous results.

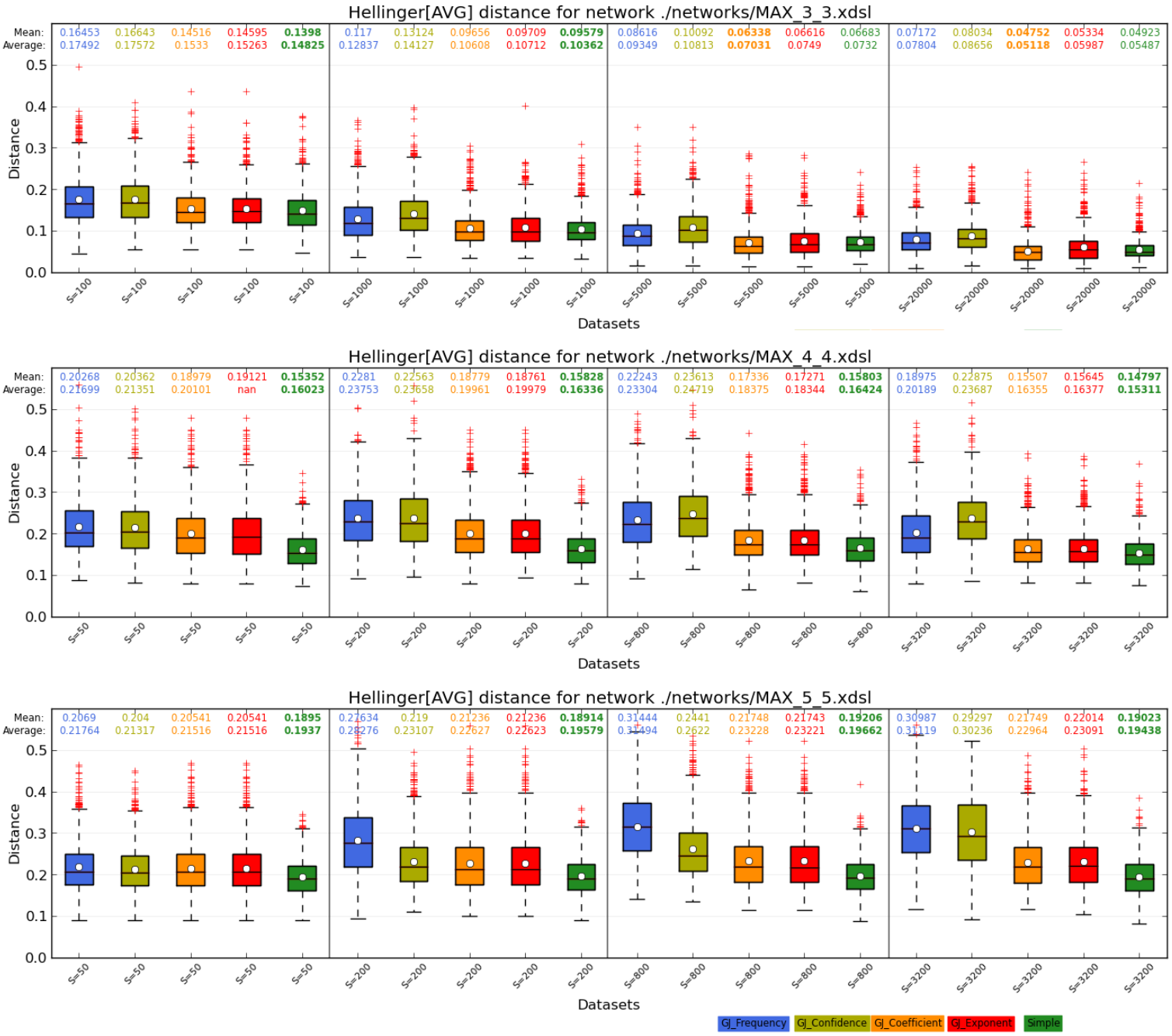

Figure 6.9: Comparison of the Gauss–Jordan based methods and the Simple Counting for the non–binary models.

Conclusion

Relatively poor performance of Gauss–Jordan based methods could be due to previously addressed problem – difficulty of handling incorrect probabilities. A similar scenario occurs here: parameters for Noisy–MAX, when obtained by means of solving of the systems of equations, may results in each of the parameters for given column within the Noisy–MAX table, to be calculated from completely different equations. Since the last value of the distribution is supplemented to 1, we compute every parameter for given column, but last. For the non–binary Noisy–MAX we cannot be certain that the distribution will add up to 1, since the sum of first $n-1$ parameters for given column, can exceed 1. In that case, we loose the means for expressing the last parameter effectively.

6.2.8 Testing the behavior to Dirichlet noising

The following experiment will test the change in the accuracy of the methods when the noise in introduced to the ideal Noisy-MAX distribution.

Tested variables

- **M** = $\{Q_{FQ}, Q_I, Q_C[\sigma = 3], Q_{Exp}[\omega = 0.3],$ Simple Counting$\}$
- **D** = $\{\text{OR}_{50}, \text{OR}_{200}, \text{OR}_{800}, \text{OR}_{3200}\}$
- **P** = $\{2, 4, 8\}$
- **K** = $\{10,000, 16, 8, 4, 2\}$
- We will use the Hellinger[AVG] and Euclidian[AVG] distances for the comparison.

Results of the experiment can be seen in Figure 6.10.

Observations

1. All methods start to perform worse as the model is further from Noisy-MAX.
2. All methods behave similarly, however, Simple Counting tends to achieve smaller variance and better mean and average when compared to Gauss–Jordan based methods.

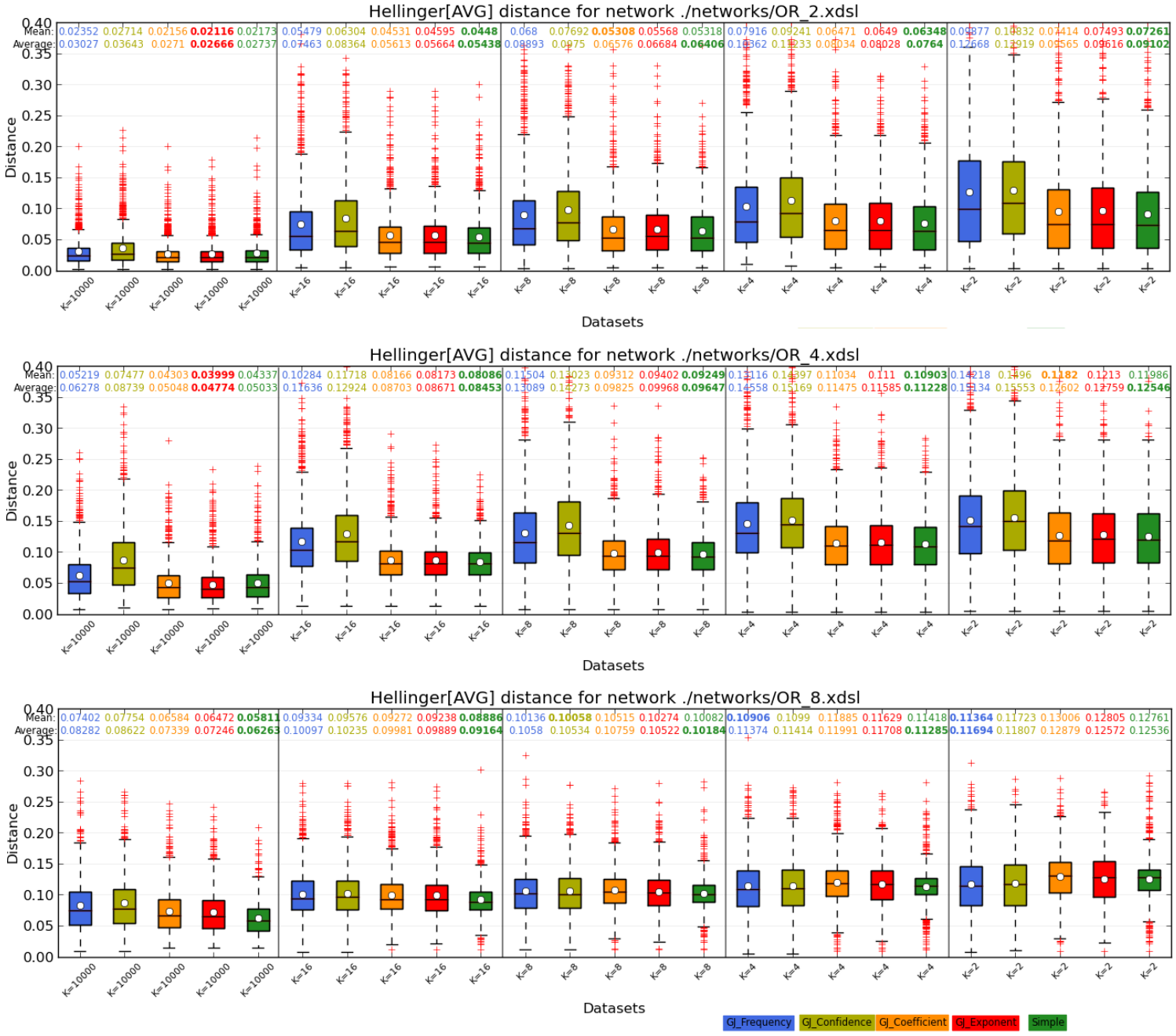

Figure 6.10: Learning accuracy for the varying distortion of the Noisy-MAX distribution (Hellinger metric).

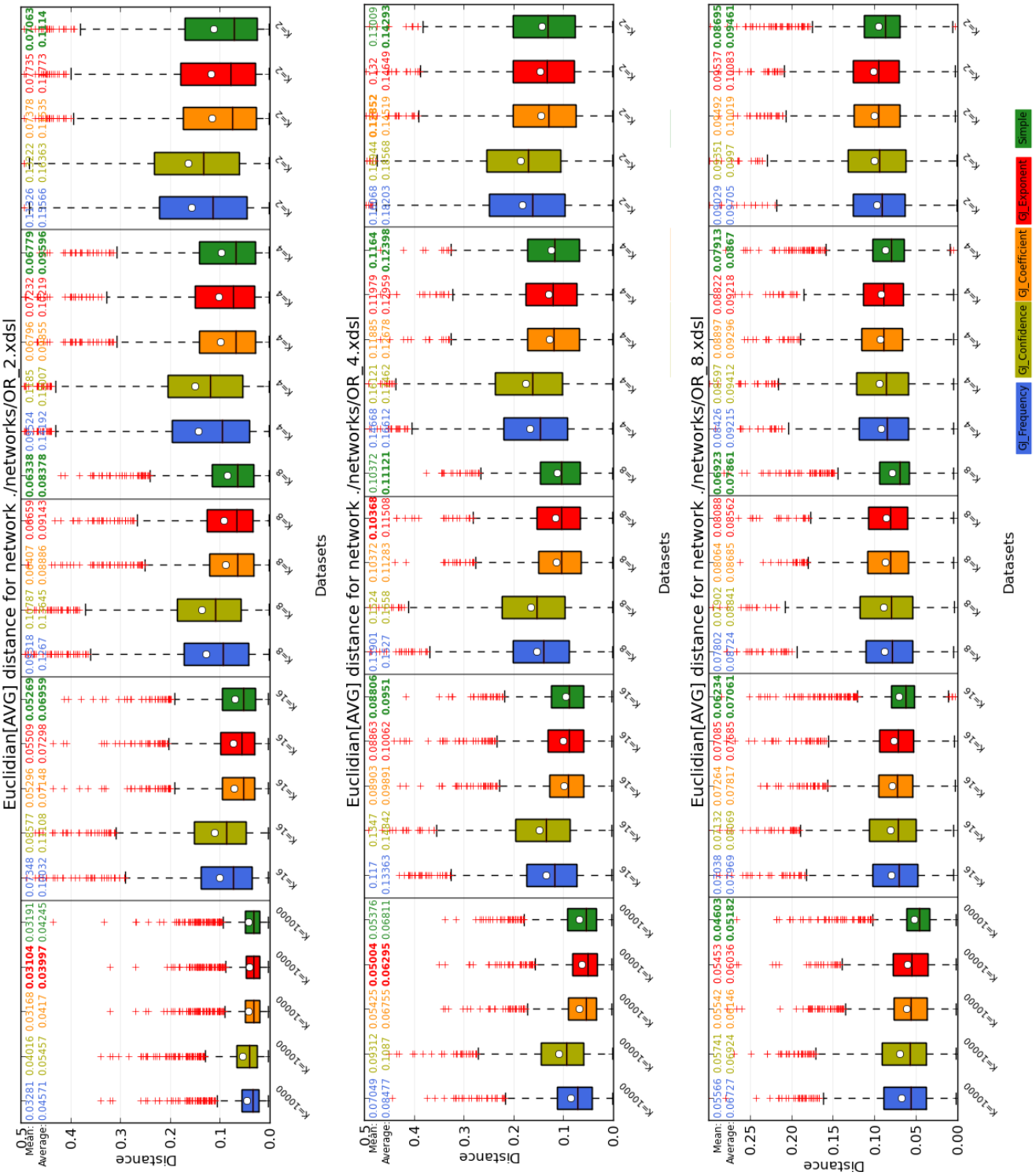

Figure 6.11: Learning accuracy for the varying distortion of the Noisy-MAX distribution (Euclidian metric).

Conclusion

Noising the original distribution does introduce some error, however Simple Counting seems to be slightly less affected by it. This is also surprising to us, since we expected the Gauss–Jordan based methods to perform better in this field.

7. Open problems

In this section we will discuss possible ways of improving of the methods, along with the yet unexplored paths for the parameter learning.

7.1 Improvement of the Confidence Interval function

Fitness function that was based on confidence interval performed way worse than we had expected. My suspicion is that it is not well fitted for a stand–alone quality function, yet it may perform better as a confidence component to any of the remaining Gauss–Jordan based methods. Main strength of this quality measure is the fact that it is strongly backed up by many statistical applications, which fits very well to the domain of our problem.

7.2 Exploration of zero vectors as a formal optimisation problem

I believe that the concept of exploration of zero vectors, as introduced in Section 4.2.2, may yield the best results so far when explored properly. Since it is built on previous concepts of quality functions (which were my main focus for the majority of the time spent on basic research), it was not explored throughly yet. We would like to propose an idea for expressing our exploratory efforts as a formal optimisation problem. Initial definition of linear combination of zero vectors was as follows:

$$\begin{aligned} s &= [1, \alpha_1, \alpha_2, \ldots, \alpha_n] \cdot [s_0, z_1, z_2, \ldots, z_n] = \\ &= s_0 + \alpha_1 \cdot z_1 + \alpha_2 \cdot z_2 + \cdots + \alpha_n \cdot z_n \ . \end{aligned} \tag{7.1}$$

Standard form of the linear programming optimization problem can be defined as fol-

lows:

Minimize:

$$f(x_1, x_2, \ldots, x_n) = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots c_n + x_n \ ,$$

Constained to:

$$\begin{aligned} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots a_{1n} \cdot x_n &\leq b_1 \ . \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots a_{2n} \cdot x_n &\leq b_2 \ . \\ &\vdots \\ a_{k1} \cdot x_1 + a_{k2} \cdot x_2 + \ldots a_{kn} \cdot x_n &\leq b_k \ . \end{aligned} \tag{7.2}$$

With non-zero parameters:

$$x_1, x_2, \ldots, x_n > 0 \ ,$$

Let us show some similarities between two definitions:

- Vector of constants: $[c_1, c_2, \ldots, c_n] \sim [s_0, z_1, z_2, \ldots, z_n]$
- Vector **x**: $[x_1, x_2, \ldots, x_n] \sim [1, \alpha_1, \alpha_2, \ldots, \alpha_n]$
- Optimised function: $f(\vec{x}) = \vec{c} \cdot \vec{x} \sim f(\vec{\alpha}) = \vec{z} \cdot \vec{\alpha}$

In standard form of an optimisation problem, $\vec{c}$ is a vector of constant scalars over the field $\mathbb{R}$. In our problem, $\vec{z}$ a vector of zero–vectors over the linear space $\mathbb{V}_{\mathbb{R}}$. If we were to propose good constraints to our problem, we can try to apply the paradigm of linear programming to the problem, or even propose a counterpart algorithm, e.g. Simplex.

Efficient solving of our zero–vectors problem would aid our naive exploration approach, in and of itself, will result in improving the learning of the Noisy-MAX parameters.

Summary

In this document, we have presented a theoretical background for the possible methods of learning the Noisy–MAX parameters from data. We have proposed few possible variants of the core method, which we confronted with the Simple Counting method, and the approach found in SMILE$^{\odot}$. We tested the accuracy of the parameter learning using generated data. We have verified some of the conjectures and ideas that emerged throughout the basic research using simple statistical indicators and the distance functions for the probability distributions, commonly found in literature – Euclidian and Hellinger distance.

Surprisingly enough, the results turned out to disprove few beliefs we originally had when we approached this research. First of all, solving Noisy–MAX equation does allow for the improved learning of the canonical parameters. Section 6.2.5 shows that it emerges in cases other than previously forecasted, yet there are scenarios where employing Gauss–Jordan elimination is not a futile endeavour. We expected Gauss–Jordan to achieve better results, especially when compared to Simple Counting, yet we are certain that the space of possible solutions has not yet been exhausted.

Both the Gauss–Jordan based methods and the Simple Counting, perform worse than the solution implemented in SMILE$^{\odot}$; this is not the case however, when the number of records is relatively small to the number of parents 6.2.6. This may pose as a possible improvement to the SMILE$^{\odot}$software.

We did not test the quality with which the methods learn parameters from the real data. This is probably the next step in order to examine the possible usability of Gauss–Jordan based methods in real applications. Additionally, neither did we verify the quality of classification of thus learned Noisy–MAX models. This may be particularly important for the future research, as it characterizes the main application of machine learning models – effectiveness of classification and prognostics.

Bibliography

[1] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[2] F. Javier Díez and Marek J. Druzdzel. Canonical probabilistic models for knowledge engineering. Technical report, UNED, Madrid, Spain, 2006.

[3] Agnieszka Oniśko, Marek J. Druzdzel, and Hanna Wasyluk. Learning Bayesian network parameters from small data sets: Application of Noisy-OR gates. In *Working notes on the European Conference on Artificial Intelligence (ECAI) Workshop Bayesian and Causal Networks: From Inference to Data Mining*, August 22 2000.

[4] Adam Zagorecki, Mark Voortman, and Marek J. Druzdzel. Decomposing local probability distributions in Bayesian networks for improved inference and parameter learning. In Geoff Sutcliffe and Randy Goebel, editors, *Recent Advances in Artificial Intelligence: Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS–2006)*, pages 860–865, Menlo Park, CA, 2006. AAAI Press.

[5] David Heckerman. A tutorial on learning with Bayesian networks. In Michael I. Jordan, editor, *Learning in Graphical Models*. The MIT Press, Cambridge, Massachusetts, 1998. Available on web at ftp://ftp.research.microsoft.com/pub/tr/TR-95-06.ps.

[6] Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. In Michael I. Jordan, editor, *Learning and Inference in Graphical Models*, pages 421–459. The MIT Press, Cambridge, MA, 1999.

[7] Decision Systems Laboratory at the University of Pittsburgh. GeNIe and SMILE software packages. http://genie.sis.pitt.edu/, 1998-2013. [Online; accessed 11-June-2013].

[8] GNU Project Free Software Foundation. GNU Scientific Library. http://www.gnu.org/software/gsl/. [Online; accessed 11-June-2013].

[9] Python Software Foundation. Python Programming Language. http://www.python.org/. [Online; accessed 11-June-2013].

[10] SciPy. Scientific Computing Tools for Python. http://www.scipy.org/. [Online; accessed 11-June-2013].

List of Tables

List of Figures